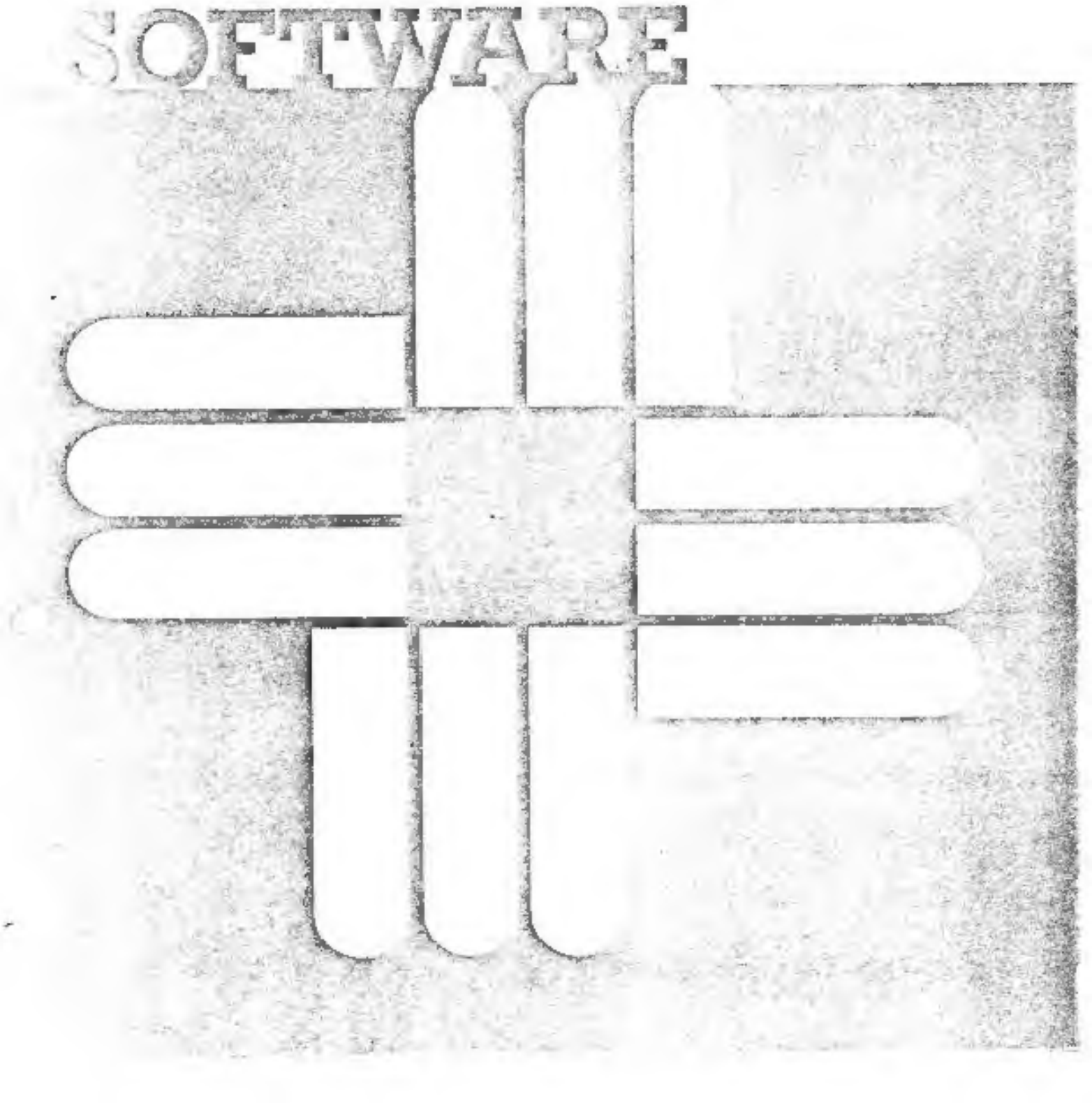
MODCOMP
MODULAR COMPUTER SYSTEMS, INC

MAX IV and MAX 32
Programmer's Reference Manual

Library Update



MANUAL HISTORY

Manual Order Number: 211-804008-H01

Title: MAX IV AND MAX 32 PROGRAMMER'S REFERENCE MANUAL, LIBRARY UPDATE

Product Number: 611599-108H.0 and 612599-108A.0

Revision Level	Date Issued	Description
---	07/76	Initial issue.
F00	05/78	Reissue (F.0).
F01	02/79	Reissue (F.1).
G00	08/81	Reissue (G.0).
G01	10/82	Reissue.
G02	04/83	Reissue (G.1).
G03	04/84	Reissue. MAX III/IV Library Update (600599-108G.1).
H00	09/84	Reissue (H.0). Supports MAX IV Library Update (611599-108H.0).
H01	05/85	Reissue (H.0 and A.0). Supports MAX IV LIB (611599-108H.0) and MAX 32 LIB32 (612599-108).

Contents subject to change without notice.

PROPRIETARY INFORMATION

THIS MANUAL CONTAINS CONFIDENTIAL PROPRIETARY INFORMATION PROTECTED BY SOFTWARE PROPERTY RIGHTS AND IS MADE AVAILABLE UPON THE CONDITION THAT THE SOFTWARE CONTAINED HEREIN WILL BE HELD IN ABSOLUTE CONFIDENCE AND MAY NOT BE DISCLOSED IN WHOLE OR IN PART TO OTHERS WITHOUT THE PRIOR WRITTEN PERMISSION OF MODULAR COMPUTER SYSTEMS INC

FOR GOVERNMENT USE THE FOLLOWING SHALL APPLY

RESTRICTED RIGHTS LEGEND

USE DUPLICATION OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN PARAGRAPH (b) (3) (B) OF THE RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE in DART 104 (b)(3) OR SUCH EQUIVALENT FAR NASPA AND/OR DOEPR CLAUSE AS MAY BE APPLICABLE)

MODULAR COMPUTER SYSTEMS INC
1650 WEST MCNAB ROAD
FORT LAUDERDALE FL 33309

PREFACE

Audience
This manual is directed to programmers using MAX IV Library Update (LIB) in the development of 16-bit programs in a MAX IV Operating System environment and to programmers using MAX 32 Library Update (LIB32) in the development of 16-bit or 32-bit programs in a MAX 32 Operating System environment.

Subject
This manual contains detailed reference information and complete descriptions of the directives associated with the MAX IV and MAX 32 Library Updates.

Product Support
MAX IV Library Update can run under the MAX IV (revision H.2 or later) Operating Systems. MAX 32 Library Update can run under the MAX 32 (revision A.0 or later) Operating Systems. LIB/LIB32 uses the MAX III form of Reentrant Executive (REX) services. The directives dealing with directoried libraries require the REX services that are required by the MAX IV/32 Source Editor, particularly the MAX III/IV compatible REX service (REX,#32).

Special Symbols and Notations
A revision bar (|) located in the margin of the page in the text indicates a change to the manual. This change generally represents a technical change to the product due to product revision. A revision bar is also entered in the Table of Contents to flag the general location of changes in the text.

Related Publications
The reader is referred to the following manuals for additional information. When ordering manuals, use the Manual Order Number listed below. The latest revision (REV) will be shipped.

Manual Order Number	Manual Title
211-804002-REV	MAX IV/MAX 32 NONRESIDENT JOB CONTROL AND BATCH FACILITIES Programmer's Reference Manual (PRM)
213-804001-REV	MAX IV GENERAL OPERATING SYSTEM System Guide Manual (SGM)
213-830001-REV	MAX 32 GENERAL OPERATING SYSTEM System Guide Manual (SGM)
211-804010-REV	MAX IV AND MAX 32 LINK EDITOR (M4EDI & LNK32) Programmer's Reference Manual (PRM)
210-804001-REV	MODCOMP ASSEMBLERS Language Reference Manual (LRM)
213-804005-REV	MAX IV BASIC I/O SYSTEM System Guide Manual (SGM)

213-838005-REV	MAX 32 BASIC I/O SYSTEM System Guide Manual (SGM)
213-804007-REV	MAX IV DATA STORAGE DEVICE HANDLERS System Guide Manual (SGM)
213-838007-REV	MAX 32 DATA STORAGE DEVICE HANDLERS System Guide Manual (SGM)

MODCOMP Product Training

MODCOMP's Education Services department provides training courses on many hardware and software products at our Training Center in Fort Lauderdale, Florida. On-site courses at customer sites can also be arranged. For more information about the courses offered by Education Services, contact the MODCOMP Training Registrar.

LIB REVISION H.0 SUMMARY

Revision H.0 of LIB accepts single character input for the following directives:

A	for	ADD
C	for	COPY
E	for	EXIT
L	for	LIST

The following directives were added to LIB in Revision H.0:

PROMPT - enables or disables the command prompt character(.).

LAE - lists all catalog name entries in a directoried library that match the name specified.

LDE - list all catalog name entries in a directoried library.

LDS - list directory summary information.

SEA - list all Common, Internal or External definitions in a sequential library that match the name specified.

SUL - list all matching Common, Internal or External definitions in a directoried library that match the name specified.

XREF - create a cross-reference listing of all object modules on a sequential library. The listing includes Common, Internal and External references.

LIB32 REVISION A.0 SUMMARY

The LIB32 initial issue (revision A.0) contains all the features of LIB revision H.0.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
OVERVIEW OF MAX IV/MAX 32 LIBRARY UPDATE

The MAX IV Library Update (LIB) and MAX 32 Library Update (LIB32) are system processors that build and maintain libraries of object modules. LIB and LIB32 process both directored and sequential libraries. A directored library is a collection of object modules accessed through a directory. A sequential library is a collection of object modules accessed sequentially. LIB runs under the MAX IV Operating System and MAX 32 Operating System and processes 16-bit object modules. LIB32 runs under the MAX 32 Operating System as a privileged Job Control overlay and processes 32-bit object modules. From now on the terms "object modules", "libraries", and "Library Update" will refer to both 16-bit and 32-bit object modules unless a distinction is made. The user must access 16-bit libraries with LIB and 32-bit libraries with LIB32. The user will get error messages if an attempt is made to access a 32-bit library with LIB or a 16-bit library with LIB32. See Appendix A for error messages.

To appreciate the role of MAX IV and MAX 32 Library Update, a discussion of the typical workflow of program development follows. Refer to Figure 1-1.

A program written in a computer language is converted to an object module by the language compiler. If the program is written in Assembly language and is entirely self contained, the object module can be converted into a loadable task image and executed. If, however, the program references external routines either explicitly by user reference or implicitly by the generated code from a language compiler, the link edit phase of task building must obtain these routines and include them in the task image. A collection of object modules to be scanned by a link editor is called a library.

The link editor may need to scan several libraries, for example, a library of user-written routines for inclusion with the program and a library of language-support routines. The MAX IV Link Editor supports 16-bit libraries and the LINK32 Link Editor supports 32-bit libraries. Another editor, the SYSTEM Editor (SED), supports only 16-bit libraries.

Library Update (LIB and LIB32) programs, normally executed as a background overlay of Job Control (BATCH), create and maintain these libraries of object modules in either a sequential or a directored format. LIB/LIB32 reads command directives and acts on them. Some directives set the operational mode of subsequent directives; some directives move object modules to and from libraries; other directives perform utility functions. LIB/LIB32 processes modules of binary object records in MOUL/OMP standard binary format only. The inputs to LIB/LIB32 are normally the unlinked object modules created by an assembler or a high-level language compiler.

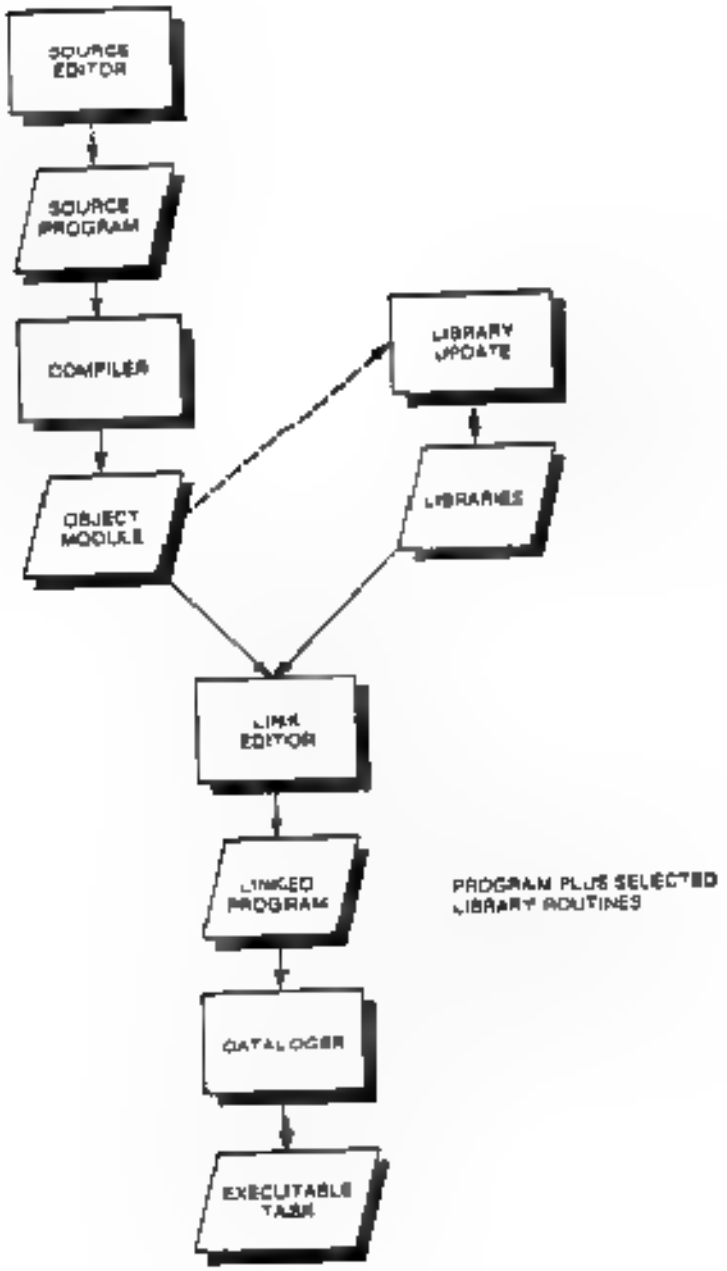


Figure 1-1. Workflow of the Stages to Build an Executable Task

CHAPTER 2
SUMMARY OF DIRECTIVES

Directives are commands used to initiate the functions within a system processor and to convey information needed by these functions. A directive consists of a directive name and parameters. The directive name initiates the function. The parameters are arguments that convey needed information such as file names, key words, and numeric values.

This chapter summarizes the directives of the Library Update (LIB and LIB32) System Processor. The directives are presented in the following functional groups:

- Job Control Directives
- Mode Directives
- Directioned Library Directives
- Sequential Library Directives

Job Control Directives used with LIB/LIB32 perform the same functions as in Job Control. Mode Directives are directives that place LIB/LIB32 into various modes of operation that affect the performance of other directives. Directory Library Directives are directives that manipulate directoried libraries, that is, object module libraries accessed through a directory. Sequential Library Directives are directives that manipulate sequential libraries, that is, object module libraries without directories.

2.1 JOB CONTROL DIRECTIVES

The following directives are equivalent to the Job Control directives of the same name. In addition, the ASSIGN, REWIND, and WEOF Directives reset program sequence numbers.

DIRECTIVE	SUMMARY
ACTION	Informs system operator of a procedure that must be performed and places LIB/LIB32 in a hold state (same as PAUSE).
ASSIGN	Causes one or more logical files of the host batch processing task to be assigned or to revert to default assignments.
AVFILE	Advances the logical file a specified number of file-mark records.
AVRECORD	Advances the logical file a specified number of physical records.
BKFILE	Moves the logical file backward a specified number of file-mark records.
BKRECORD	Moves the logical file backward a specified number of physical records.
EXIT	Terminates execution of LIB/LIB32 and returns control to Job Control.
$NOP	Can be used in a job stream or procedure to insert a user's comment record.
NOTE	Informs system operator without entering a hold.

PAUSE	Informs system operator and places LIB/LIB32 in a hold state (same as ACTION).
REWIND	Positions a file/device at beginning-of-medium.
WEOF	Writes an end-of-file mark onto a selected logical file.

2.2 MODE DIRECTIVES

The following directives place LIB/LIB32 into various modes, which affect the performance of subsequent directives.

DIRECTIVE	SUMMARY
FILE	Causes file-mark records to be copied from logical file BI to logical file BO during library updates. This mode is the default.
FUNCTION	Causes Library Update to verify the Assembler function codes of binary object modules in both sequential and directoried libraries. This mode is the default.
HOLD	Causes Library Update to enter the HOLD state before initializing a directory. This mode is the default.
LALL	Causes program names, internals, externals, common names and module sizes to be printed when a module is processed. This mode is the default.
LNAME	Causes only the program name to be listed when a module is processed.
LNS	Causes only the program name and size to be listed when a module is processed.
NFUNCTION	Suppresses the FUNCTION mode.
NOFILE	Suppresses the copying of file-mark records from BI to BO.
NOHOLD	Causes Library Update to initialize a directory without entering the HOLD state. Use with caution.
NOVERIFY	Suppresses the VERIFY mode and, by implication, the FUNCTION mode as well.
VERIFY	Causes checksums, sequence numbers, and function codes to be verified in the module being processed. This mode is the default.

2.3 DIRECTORIED LIBRARY DIRECTIVES

The following directives maintain directoried libraries on logical file UL. For each object module in a directoried library, there is an entry in the directory.

DIRECTIVE	SUMMARY
CATALOG	Copies a new module into the directoried library and creates a new directory entry for it.
INITIALIZE	Builds the directory sectors on logical file UL.
LDIR	Lists and validates the directory of logical file UL.
RECATALOG	Removes the directory entry for an existing directoried library object module, copies a new object module, and updates the directory.
REMOVE	Removes a directory entry, effectively removing the object module as well
RENAME	Changes the catalog name in the directory entry of an object module.
RESTORE	Copies a previously saved directoried library or converts a sequential library into a directoried library.
SAVE	Copies a directoried library for storage or converts a directoried library into a sequential library.
SQUEEZE	Compresses the directory entries and object modules in a directoried library to remove fragmentation of unused disc space.
TEST	Checks the integrity of object modules after a SAVE or SQUEEZE operation.
USE	Specifies a directoried library identifier.

The following sequential library directives can be used to perform operations on a directoried library:

DIRECTIVE	SUMMARY
COPY	To copy a single object module from a directoried library.
LIST	To list a single object module or the entire directoried library
POSITION	To position logical file SI to a particular object module in the directoried library.

2.4 SEQUENTIAL LIBRARY DIRECTIVES

The following directives maintain sequential libraries. When a sequential library is updated, the old library is copied from logical file SI, new modules are inserted from logical file SI, and the new sequential library is output on logical file SO.

SUMMARY

DIRECTIVE	
ADD	Copies the sequential library from logical file BI to logical file BO, until a specified object module is copied. Object modules are then inserted from logical file SI.
COPY	Copies a sequential library from logical file BI to logical file BO, until a file-mark record is read.
DELETE	Copies the sequential library up to a specified object module or range of object modules, and then skips the specified object module(s).
GET	Copies a single object module from a sequential library.
LIST	Produces a listing of the object modules of a sequential library or a number of sequential libraries.
NAME	Copies the sequential library up to a specified object module and then changes the program-name of the specified object module.
POSITION	Sets the current file pointer to the start of a specified object module.
PTAPE	(MAX IV only) Sets the number of logical records to be written as one paper tape. Can be used to split the output of a sequential library update into several small paper tapes.
RECORD	Causes the output of COPY, ADD, and REPLACE Directives to be written in records of a particular size rather than the default size for the output device.
REPLACE	Copies the sequential library up to a specified object module (or range of object modules) and replaces the specified object module (or range of object modules) with object modules from logical file SI.

CHAPTER 3
DIRECTIVES

This chapter presents each L1B/L1B32 Directive in alphabetical order. Directive descriptions include a functional description, the syntax, and examples of each.

ACTION

Display a Message and Wait for Response

The ACTION Directive displays a message on the logical file CO directed to the operator's attention. L1B/L1B32 is placed into the HOLD state, and a /RESUME Directive must be entered in order to resume execution.

SYNTAX

ACT[ION],[text]

[text] - Parameter 1 (optional) specifies a message directed to the operator's attention. If not entered, the message 'ACTION' is displayed.

EXAMPLES

ACTION SET THE WRITE-PROTECT SWITCHES FOR DISC

The message ACTION SET THE WRITE-PROTECT SWITCHES FOR DISC' is output to the CO file.

ACT MOUNT TAPE ON MT1

The message 'ACT MOUNT TAPE ON MT1' is output to the CO file

ADD

Add Object Modules to a Sequential Library

The ADD Directive adds object modules to a sequential library. The ADD Directive causes LIB/LIB32 to copy object modules from the current position of logical file BI to logical file BO until a specified object module is copied. This object module can be specified by its sequence number within the sequential library or by its program name.

LIB/LIB32 then adds the object modules from logical file SI to logical file BO until a file-mark record is encountered on logical file SI. The file-mark record is NOT written to logical file BO. LIB/LIB32 then proceeds to read the next directive from logical file CI or AI.

If a file-mark record is read before the specified object module has been found, an error message is displayed.

NOTE: Object modules on logical file BI following the specified object module can be copied to logical file BO, replaced, deleted, or added to logical file BO by subsequent COPY, REPLACE, DELETE, or ADD Directives.

SYNTAX

A[DD] sequence-number
program-name

sequence-number
program-name — Parameter is required; specifies one of the following.

sequence-number specifies the sequence-number of the last object module copied from logical file BI. The sequence-number reflects the object module's sequential position from the beginning of logical file BI. It must be a decimal integer equal to or greater than zero. If zero is specified, LIB/LIB32 bypasses the copy of object modules from logical file BI to logical file BO and proceeds with the copy of object modules from logical file SI to logical file BO.

program-name specifies the program-name of the last object module copied from logical file BI. The program-name is specified by the PGM statement in Assembly language programs.

EXAMPLES

ADD,0

LIB/LIB32 copies object modules from logical file SI to logical file BO until a file-mark record is encountered on logical file SI.

ADD 12

LIB/LIB32 copies object modules from logical file BI to logical file BO until the 12th object module in the sequential library is copied. LIB/LIB32 then proceeds to copy object modules from logical file SI until a file-mark record is encountered.

ADD PROG8

LIB/LIB32 copies object modules from logical file BI to logical file BO until the object module named PROG8 is copied. LIB/LIB32 then proceeds to copy object modules from logical file SI to logical file BO until a file-mark record is encountered.

ASSIGN

Assign Logical Files to Logical Devices or Other Logical Files

The ASSIGN Directive is used to assign a logical file(s) to a logical device(s) or to a logical file(s). If a logical file is assigned to itself (for example ASS LO:,O), it is assigned to its default assignment in the system. A file that has no default assignment becomes vacant as a result of this.

A new file of the host batch processing task can be opened by ASSIGN if that task has vacant file assignments in its File Assign Table. The logfilename name that is not already in the table is opened if a vacancy exists. Such new files are not given a default assignment.

If a logical file is assigned to a device, then the File Position Index in the File Assign Table is reset to 0. If the file is assigned to another file, then its File Position Index becomes that of the other logical file.

Further information about logical files can be obtained from the MAX IV or MAX 32 BASIC I/O SYSTEM, System Guide Manual.

SYNTAX

ASS[IGN] logfilename logfiletwo
devicename

logfilename - Parameter 1 (required) specifies the local logical device file of the host batch task to be linked to a device or other logical file.

logfiletwo
devicename - Parameter 2 (required) can be one of the following:

logfiletwo specifies the name of another logical file already assigned and can be a global file name.

devicename specifies the name of a system device to be linked to the logical file being assigned.

EXAMPLES

ASSIGN SI=CR
Assign source input, the SI file, to the card reader device.

ASS SO MT1
Assign source output, the SO file, to magnetic tape drive 1.

Advance the Logical File a Specified Number of File-Mark Records

The Advance File (AVFILE) directive is used to advance a device medium until a number of file-mark records are read. The File Position Index in the File Assign Table is incremented by 1 for each physical record encountered. The way specific devices respond to a single AVFILE Directive is detailed in the MAX IV or MAX 32 BASIC I/O SYSTEM, System Guide Manual.

SYNTAX

AVF[ILE] filename [integer]

filename - Parameter 1 (required) specifies the name of the logical file to be advanced.

[integer] Parameter 2 (optional) specifies the number of file-mark records. If not specified, a value of 1 is assumed.

EXAMPLES

AVFILE SI
Advance logical file SI one file-mark record.

AVF SI,2
Advance logical file SI two file-mark records.

AVRECORD

Advance the Logical File a Specified Number of Physical Records

The Advance Record (AVRECORD) directive advances a device medium until a number of physical records are skipped. The File Position Index in the File Assign Table is incremented by 1 for each physical record encountered.

The way individual devices respond to a single AVRECORD directive is specified in the MAX IV or MAX 32 BASIC I/O SYSTEM, System Guide Manual.

If the end-of-medium position is reached before the specified number of physical records is encountered, the following message is displayed on the logical file CO which is usually assigned to the terminal:

END OF MEDIA

The specified logical file remains positioned beyond the last physical record until another file positioning directive is entered.

SYNTAX

AVR[ECORD] filename [integer]

filename – Parameter 1 (required) specifies the name of the logical file to be advanced.

[integer] – Parameter 2 (optional) specifies the number of physical records. If not specified, a value of 1 is assumed.

EXAMPLES

AVRECORD SI
Advance logical file SI one physical record.

AVR SI,2
Advance logical file SI two physical records.

Move the Logical File Backward a Number of File-Mark Records

The Back File (BKFILE) Directive moves a device medium in the reverse direction a specified number of file-mark records. If the device reaches its beginning-of-medium point on before the specified number of file-mark records have been bypassed, the remaining backspace file requests cause no additional movement of the device.

The File Position Index in the File Assign Table is decremented by 1 for each physical record encountered.

The way particular devices respond to a single BKFILE Directive is given in the MAX IV or MAX 32 BASIC I/O SYSTEM, System Guide Manual.

SYNTAX

BKF[ILE] filename [integer]

filename - Parameter 1 (required) specifies the name of the logical file to be reversed.

[integer] - Parameter 2 (optional) specifies the number of file-mark records. If not specified, a value of 1 is assumed.

EXAMPLES

BKFILE BI
Move logical file BI backward one file-mark record.

BKF BI,2
Move logical file BI backward two file-mark records.

BKRECORD

Move the Logical File Backward a Number of Physical Records

The Back Record (BKRECORD) Directive moves a device medium in the reverse direction a specified number of physical records. If the particular device cannot perform such an operation, no operation is performed.

If the device reaches its beginning-of-medium position before the count is satisfied, the remaining backspace commands do not cause further motion.

The File Position Index in the File Assign Table is decremented by 1 for each physical record encountered.

The way various devices respond to a single BKRECORD Directive is described in the MAX IV or MAX 32 BASIC I/O SYSTEM, System Guide Manual.

SYNTAX

BKR[ECORD] filename [integer]

filename - Parameter 1 (required) specifies the name of the logical file to be reversed.

[integer] - Parameter 2 (optional) specifies the number of physical records. If not specified, a value of 1 is assumed.

EXAMPLES

BKRECORD B:
Move logical file B: backward one physical record.

BKR S:1,2
Move logical file S: backward two physical records.

Add a Module to a Directoried Library and Update the Directory

The CATALOG Directive adds an object module to a directoried library and updates the directory. LIB/LIB32 copies an object module from logical file SI to logical file UL and records it in the directory. The object module being cataloged must not be null. The object module is copied into the largest space available.

Unless the NOVERIFY or NFUNCTION Mode has been specified, LIB/LIB32 verifies the sequence numbers, checksums, and function codes. LIB/LIB32 lists object module information in the current listing mode on the logical file LO. Refer to the NOVERIFY and NFUNCTION Directives.

If a CATALOG Directive is rejected, the position of logical file SI is undefined.

SYNTAX

CAT[ALOG] [catalog-name] [identification]

[catalog-name] - Parameter 1 (optional) specifies the catalog name of the object module. It must begin with an alpha character and must not exceed eight characters. If not specified LIB/LIB32 uses the program name of the object module being copied as the catalog name of the object module.

[identification] - Parameter 2 (optional) specifies an additional means of identifying the object module. This identification appears next to the object module catalog name on a directory listing. If not specified, the date and time when the object module was cataloged is recorded in the directory. If specified, the identification must not exceed eight characters and must begin with an alpha character.

EXAMPLES

CATALOG PROG1 SMITH

LIB/LIB32 copies the next object module from SI to UL and catalogs it under the name PROG1. The identification SMITH is also entered into the directory.

CAT PROG1

LIB/LIB32 copies the next object module from SI to UL and catalogs it under the name PROG1. The date and time are entered into the identification field in the directory.

COPY

Copy Sequential Library Object Modules from BI to BO
or
Copy One Directoried Library Object Module from JL to BO

The COPY Directive has two functions:

- Updating a sequential library by copying object modules from logical file BI to logical file BO.
- Copying a single object module in a directoried library from logical file JL to logical file BO.

When updating a sequential library, the COPY Directive causes LIB/LIB32 to copy all object modules from logical file BI to logical file BO until the first file-mark record or the specified number of file-mark records are read from BI. The copy begins at the present positions of BI and BO. The file-mark record(s) that is read from logical file BI is written to logical file BO, unless the NOFILE Mode is set. (Refer to the NOFILE Directive.) LIB/LIB32 proceeds to read the next directive.

When copying a single object module from a library directory, logical file BI must be assigned to logical file JL and JL must be POSITIONed to the object module to be copied. The COPY Directive causes LIB/LIB32 to copy the object module to BO with a following file-mark record unless the NOFILE Mode is set.

SYNTAX

C[OPY]] [integer]

[integer] - Parameter 1 (optional) specifies the number of file-mark records to be read by LIB/LIB32 during a copy of sequential library object modules. The copy proceeds until the specified number of file-mark records are read or until a double file-mark is read. A double file-mark consists of two consecutive file-mark records with no intervening data. If not specified, the default value is 2.

EXAMPLES

COPY 25
LIB/LIB32 copies all object modules in 25 sequential libraries from logical file BI to logical file BO.

COPY
LIB/LIB32 copies one object module in a directory library from logical file JL to logical file BO under the following conditions. Logical file BI must be assigned to logical file JL and JL must be POSITIONed to the object module to be copied.

Otherwise, LIB/LIB32 copies all object modules in one sequential library from BI to BO.

Remove a Module or a Range of Modules from a Sequential Library

The DELETE D rect ve causes LIB/LIB32 to remove an object module or a range of object modules from a sequential library. The object module(s being de eted can be spec fied by sequence-number with n the sequential library or by program name. LIB/LIB32 reads the sequential library from logical f. e BI and wr tes it to logical file BO. Figure 3-1 illustrates LIB's use of logical f les with the DELETE D rective.

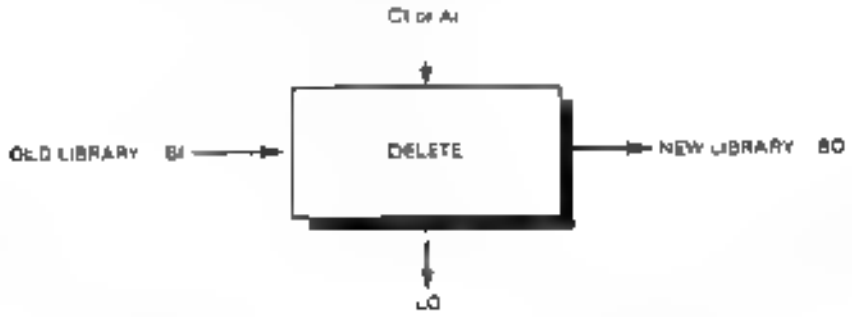


Figure 3–1. Use of Logical Files in the DELETE Directive

If a file-mark record s read before the object modules designated for delet on have been found, LIB/LIB32 proceeds to read the next direct ve. The next directive is normally a COPY or a WE OF BO D rect ve.

The corresponding directive used to remove object modu es from directoried l braries s REMOVE.

There are two stages in a delete operat on.

1) Modules are read from BI and cop ed to BO up to, but not including the first module to be de eted.

2) Modules up to and nclud ng the last module to be deleted are sk pped over on the BI file. BI s posit oned after the ast module deleted. f the last module is specified by number and is before the first module in the library, stage two w ll not be performed and an error message w ll be generated.

SYNTAX

DEL[ETE] sequence-number-1 [sequence-number-2]
program-name-1 [program-name-2]

sequence-number-1
program-name-1

- Parameter 1 (required) specifies one of the following:

sequence-number-1 specifies the sequence number of the object module to be deleted. The sequence-number reflects the object module's sequential position from the beginning of logical file 81. When used with parameter 2, sequence-number-1 specifies the sequence-number of the first object module in a range of object modules. Sequence-number-1 must be a decimal number greater than zero.

program-name-1 specifies the program name of the object module to be deleted. When used with parameter 2, program-name-1 specifies the program name of the first object module in a range of object modules.

[sequence-number-2] or
[program-name-2]

Parameter 2 (optional) specifies one of the following:

sequence-number-2 specifies the sequence- number of the last object module in a range of object modules to be deleted. Sequence-number-2 must be a decimal number greater than sequence-number-1. If sequence-number-1 is greater than sequence-number-2, LIB/LIB$2 sends an error message to the operator.

program-name-2 specifies the program name of the last object module in a range of object modules to be deleted. Program-name-1 must precede program-name-2 in the sequential library being updated.

EXAMPLES

DEL,7
The 7th object module in the sequential library is deleted.

DEL 9,12
The 9th through the 12th object modules in the sequential library are deleted.

DEL PROG14
The object module with the program name PROG14 is deleted from the sequential library.

DEL FILEMAN,RECMAN
All consecutive object modules in the range FILEMAN through RECMAN are deleted from the sequential library.

End LIB/LIB32 and Return to Job Control

The EXIT Directive causes the operating system to load Job Control and transfer control to it.

SYNTAX

```
E[XIT]
```

EXAMPLES

EXIT
Exit from LIB/LIB32 and return to Job Control.

E
Exit from LIB/LIB32 and return to Job Control

FILE

Enter the "Copy File-Mark Records" Mode

The FILE Directive places LIB/LIB32 in the mode in which the COPY Directive copies file-mark records onto logical file 80 every time a file-mark record is read on logical file 81. The FILE Directive is used when updating sequential libraries. The FILE Mode is the default mode when LIB.LIB32 is executed. This mode is turned off by the NOFILE Directive.

SYNTAX

FIL[F]

EXAMPLES

FILE
The FILE Mode is set

FIL
The FILE Mode is set.

Enter the "Verify Assembler Function Codes" Mode

The FUNCTION Directive places LIB/LIB32 in the mode in which LIB/LIB32 verifies Assembler function codes. The FUNCTION Mode is the default mode when LIB/LIB32 is executed. The mode is turned off by the NFUNCTION Directive.

SYNTAX

FUN[CTION]

EXAMPLES

FUNCTION
The FUNCTION Mode is set.

FUN
The FUNCTION Mode is set.

Write a Module From a Sequential Library to Logical File 80

The GET directive causes LIB/LIB32 to extract an object module from a sequential library. The object module can be specified by sequence-number within the sequential library or by program-name. LIB/LIB32 reads the specified object module from the sequential library on logical file 81 and writes the object module to logical file 80. Figure 3-2 illustrates the LIB/LIB32 use of logical files with the GET Directive.

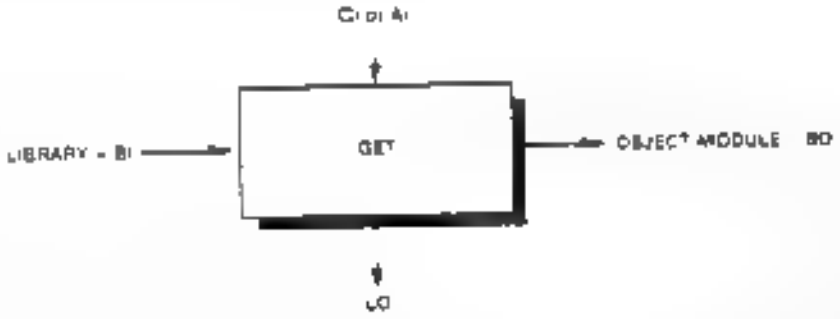


Figure 3-2. Use of Logical Files in the GET Directive

LIB/LIB32 sequentially searches the sequential library from its current position on the logical file 81. The object module specified by sequence number or by program name is written to logical file 80. LIB/LIB32 then proceeds to read the next directive.

If a file-mark record is read before the specified object module has been found, an error message is displayed and LIB/LIB32 proceeds to read the next directive.

SYNTAX

GET sequence-number
program-name

sequence-number
program-name - Parameter 1 (required) must specify one of the following:

sequence-number identifies the object module to be copied to logical file 80. The sequence-number reflects the object module's sequential position from the beginning of logical file 81. It must be a decimal number greater than zero.

program-name specifies the program name of the object module to be copied to logical file 80.

EXAMPLES

GET 5

LIB/LIB32 searches the sequential library from its current position on logical file 81 for the 5th object module in the library and, if found, copies it to logical file 80.

GET PROG34

LIB/LIB32 searches the sequential library from its current position on logical file 81 for the object module under the program name PROG34 and, if found, copies it to logical file 80.

HOLD

Enter the Hold State Before Initializing a Directory

The HOLD directive places LIB/LIB32 in the mode in which a HOLD will be performed before initializing a directoried library. The HOLD mode is the default mode when LIB/LIB32 is entered. Refer to the NOHOLD directive.

SYNTAX

HOL[D]

EXAMPLE

```
HOLD
    The HOLD mode is set.
```

Builds the Directory Sectors on the Logical File UL
***** USE WITH CAUTION *****

The INITIALIZE Directive builds the directory sectors on the logical file UL to enable subsequent insertion of directory entries. LIB/L1832 enters the HOLD state and displays a warning before the initialization. If resumed, LIB/L1832 builds a new directory even if a directory currently exists on logical file UL. ALL PREVIOUS DATA ON LOGICAL FILE UL IS LOST IF NOT BACKED-UP. Logical file UL must be assigned to a disc partition.

CAUTION

To avoid loss of data, back-up logical file UL before initialization.

SYNTAX

INI[TIALIZE] [integer] [identification]

[integer] - Parameter 1 (optional) specifies the number of object module entries that the directory will need to hold. LIB/L1832 uses this number to determine how many directory sectors to build. If not specified, one sector is used.

[identification] - Parameter 2 (optional) specifies the library identifier of the directoried library to which logical file UL is assigned. The identifier must begin with an alpha character and must not exceed eight characters in length.

EXAMPLES

INITIALIZE
LIB/L1832 builds the directory sectors for logical file UL. One sector is utilized.

INI 50 YANK
LIB/L1832 builds enough directory sectors for logical file UL to hold 50 directory entries. The directoried library identifier is YANK.

LAE

List All Matching Catalog Name Entries in Directoried Library

The List All Entries (LAE) directive causes LIB/LIB32 to list all occurrences of the specified catalog name. LIB/LIB32 searches the directory of the UL file for catalog names that match the name specified as parameter 1. The matches are made ignoring the character positions specified as plus signs ('+'). When a match is found, LIB/LIB32 lists the catalog name.

LIB/LIB32 checks the data integrity of matched modules if the TEST parameter is specified and UL is assigned to a disc. TEST is only valid for disc files.

SYNTAX

LAE name [[TES]T]

name - Parameter 1 (required) specifies the catalog name to be matched with the directory entries. The name consists of one to eight characters, each of which is either a valid character that can be used in a name, or a plus sign. The plus sign ('+') is used as a wild card character. Matches are made ignoring the character positions held by any plus signs. When a match is made, the catalog name is listed.

[[TES]T] - Parameter 2 (optional) is the keyword TEST. If TEST is specified and UL is assigned to a disc, LIB/LIB32 checks the data integrity of each matched module.

EXAMPLES

LAE +++NEW
List all six-character names in the UL directory that have "NEW" as the fourth, fifth and sixth characters of the catalog name.

LAE F$+++
List all five-character or less catalog names in the UL directory that have "F$" as the first two characters of the catalog name.

LAE X++++++++
List all program names in the UL directory that have "X" as the first character of the catalog name.

Enter the "List All Attributes" Mode

The .list ALL (.ALL) Directive places LIB/LIB32 in the mode in which the following object module attributes are listed on the logical file LO.

- Program name
- Internals
- Externals
- Common names
- Size
- Counter attribute assignment

These attributes are listed whenever an ADD, CATALOG, COPY, DELETE, GET, LIST, RECATLOG, or REPLACE Directive is executed. .ALL is the default mode when LIB/LIB32 is executed. Refer to the List Name (.LNAME) and to the List Name and Size (.LNS) Directives.

Figure 3-3 illustrates a listing generated by the LIST Directive under the .ALL mode, using LIB.

```
1   P HELLO                 I HELLO  02 0000    A 0002    02 0000
    COUNTER 02     LAST 0000     BOUNDARY 0001    SIZE  0000   13
    TRANSFER ADDRESS  NONE       COMMON NONE
```

Figure 3-3. LIST Directive Output in .ALL Mode Using LIB

Figure 3-4 illustrates a listing generated by the LIST Directive under the .ALL mode, using LIB32. The comment field is a new feature in LIB32. Please refer to the PGM Directive in the MAX IV/MAX 32 Assemblers, Language Reference Manual (LRM).

```
1   P HELLO  THIS,IS,A,COMMENT
    I HELLO 02   00000000  A 0002    02 00000000
    COUNTER 02 LAST  0000000D   BOUNDARY 0001   SIZE 0000000D  13
    TRANSFER ADDRESS NONE              COMMON NONE
```

Figure 3-4. LIST Directive Output in .ALL Mode Using LIB32

SYNTAX

```
.ALL[L]
```

EXAMPLES

```
␣ALL
```

The ␣ALL Mode is set.

```
AL
```

The ␣ALL Mode is set.

List Catalog Name Entries in Directoried Library

The List Directory Entries (LDE) Directive causes LIB/LIB32 to list all catalog name entries in the directory. LIB/LIB32 reads the directory of the UL file and lists the catalog names to the logical file LO.

When the TEST parameter is specified and UL is assigned to a disc, LIB/LIB32 checks the data integrity of each object module in the UL file. TEST is only valid for disc files.

SYNTAX

LDE [[TES]T]

[[TES]T] Parameter 1 (optional) is the keyword TEST. If TEST is specified and UL is assigned to a disc, LIB/LIB32 checks the data integrity of each object module.

EXAMPLES

LDE
List the directory entries of the UL file.

LDE TEST
List the directory entries of the UL disc file and check the integrity of each object module.

.DIR

Validate the Directory and Produce a Directory Listing of UL

The List Directory (.DIR) Directive validates the directory of logical file UL and produces a directory listing of information on object modules on UL. If UL is assigned to a disc file, the following information is also listed:

- Number of entries cataloged.
- Number of directory entries used.
- Number of unused directory entries.
- Largest number of continuous unused sectors.
- Total number of unused sectors.
- Total number of sectors used.
- The warning message NEEDS TO BE COMPRESSED if applicable.

SYNTAX

.DI[R] [TEST]

[TEST] - Parameter 1 (optional) specifies that each object module on the logical file UL is read and that the sequence number and checksum of each object module is validated.

EXAMPLES

.DIR
.IB/LIB32 produces a directory listing of the object modules on logical file IL

.DIR TEST
.IB/LIB32 produces a directory listing of the object modules on logical file UL and validates the sequence numbers and checksums of all object modules.

List Directory Summary

The List Directory Summary (LDS) Directive causes LIB/LIB32 to examine the directory of the UL file. If UL: is assigned to a disc, LIB/LIB32 lists the following information:

- Number of entries cataloged
- Number of directory entries used
- Number of unused directory entries
- Largest number of contiguous unused sectors
- Total number of unused sectors
- Total number of sectors used
- Attached file names
- If unused space is fragmented, LIB/LIB32 displays the message "NEEDS TO BE COMPRESSED"

LIB/LIB32 checks each object module in the UL: file for data integrity, if the TEST parameter is specified and UL: is assigned to a disc. TEST is only valid for disc files.

SYNTAX

LDS [[TEST]T]

[[TEST]T] - Parameter 1 (optional) is the keyword TEST. If TEST is specified and UL: is assigned to a disc, LIB/LIB32 checks each object module for data integrity.

EXAMPLES

LDS
List the directory summary of the UL:.

LDS TEST
List the directory summary and check the data integrity of each module in the UL: disc file.

List the Attributes of Object Modules

The LIST Directive performs the following functions:

- Lists the attributes of object modules in a single sequential library or a number of sequential libraries.
- Lists the attributes of object modules in a single directoried library.
- Lists the attributes of a single object module within a directoried library.

List Sequential Library

When using one or more sequential libraries, the libraries are read from the logical file BI and listed on the logical file LO. If parameter 1 is included, the number of successive sequential libraries specified by parameter 1 is listed. The listing stops when a file-mark record or double file-mark records in multiple file mode, are detected. LIB/LIB32 then proceeds to read the next directive.

List Directoried Library

When using a directoried library, the attributes of all of the cataloged object modules in the library are listed on LO in the order of the directory if the logical file BI is assigned to DL and if DL is positioned at the beginning of the directoried library. Parameter 1 is not required; it is ignored if specified.

List Directoried Module

When accessing an object module in a directoried library, the logical file BI must be assigned to DL and DL must be positioned at the beginning of the object module in the library. Parameter 1 is not required; specifying parameter 1 in this case causes contiguous modules on the logical file BI to be listed on LO. These object modules may not be contiguous in the directoried library or may not be in the directoried library at all.

SYNTAX

L[IST] [integer]

[integer] - Parameter 1 (optional) specifies the number of successive sequential libraries to be accessed. If not specified, a value of one is used.

Do not specify parameter 1 when accessing a directoried library or an object module within a directoried library.

EXAMPLES

LIST,$

LIB/L B32 lists the attributes of object modules in three successive sequential libraries from logical file BI to logical file LO.

LIS

If logical file BI is assigned to UL and UL is positioned at the beginning of a directoried library, the attributes of all object modules within that directoried library are listed on LO.

If logical file BI is assigned to UL and UL is POSITIONed at the beginning of an object module in a directoried library, the attributes of that object module are listed on LO.

LNAME

Enter the "List the Program or Catalog Name" Mode

The List Name (LNAME) Directive places LIB/LIB32 in the mode in which only the program name or catalog name of object modules is listed on logical file LO whenever an ADD, CATALOG, COPY, DELETE, GET, LIST, RECATALOG, or REPLACE Directive is executed. LALL is the default mode when LIB/LIB32 is executed. Refer to the List ALL (LALL) and to the List Name and Size (LNS) Directives.

Figure 3-5 illustrates a listing generated by the LIST Directive under the LNAME Mode using LIB.

```
1  P HELLO
```

Figure 3-5. LIST Directive Output in LNAME Mode Using LIB

Figure 3-6 illustrates a listing generated by the LIST Directive under the LNAME mode using LIB32. The comment field is a new feature in LIB32. Refer to the MAXIV/MAX 32 ASSEMBLERS, Language Reference Manual (LRM) for information on the PGM Directive.

```
1  P HELLO   THIS,IS,A,COMMENT
```

Figure 3-6. LIST Directive Output in LNAME Mode Using LIB32

SYNTAX

LNAME]

EXAMPLES

LNAME
The LNAME Mode is set.

LNA
The LNAME Mode is set.

Enter the "List the Program or Catalog Name and Size" Mode

The List Name and Size (LNS) Directive places LIB/LIB32 in the mode in which the program name or catalog name and size of object modules are listed whenever an ADD, CATALOG, COPY, DELETE, GET, LIST, RECATALOG, or REPLACE Directive is executed. LALL is the default mode when LIB/LIB32 is executed. Refer to the List ALL (LALL) and to the List Name (LNAME) Directives.

Figure 3-7 illustrates a listing generated by the LIST Directive under the LNS Mode using LIB.

```
1    P HELLO
     COUNTER 02      LAST 0000      BOUNDARY 0001    SIZE 0000     13
     TRANSFER ADDRESS  NONE         COMMON NONE
```

Figure 3-7. LIST Directive Output in LNS Mode Using LIB

Figure 3-8 illustrates a listing generated by the LIST Directive under the LNS Mode using LIB32. The comment field is a new feature in LIB32. Refer to the MAX IV/MAX 32 ASSEMBLERS, Language Reference Manual (LRM) for information on the PGM Directive.

```
1    P HELLO     THIS,IS,A,COMMENT
     COUNTER 02    LAST 00000000    BOUNDARY 0001   SIZE 00000000   13
     TRANSFER ADDRESS  NONE                COMMON NONE
```

Figure 3-8. LIST Directive Output in LNS Mode Using LIB32

SYNTAX

LNS

EXAMPLE

LNS

The LNS Mode is set.

NAME

Copy a Sequential Library Module Under a New Program Name

The NAME directive copies a sequential library object module, specified by sequence number, under a new program name. When the specified sequence number is greater than the sequence number of the next object module in the sequential library, LIB/LIB32 copies object modules from logical file BI to logical file BO. When the specified sequence number is equal to the sequence number of the next object module in the sequential library, LIB/LIB32 copies this object module to BO under the program name specified in Parameter 2. LIB/LIB32 then reads the next directive.

SYNTAX

NAM[E] sequence-number program-name

sequence-number – Parameter 1 (required) specifies the sequence number of the object module in a sequential library. The sequence-number reflects the object module's sequential position from the beginning of logical file BI.

program-name – Parameter 2 (required) specifies the program name under which the object module specified by the sequence number is copied.

EXAMPLES

NAME,8,PROG56

If the sequence number of the next object module in the sequential library is equal to 8, LIB/LIB32 copies this object module to logical file BO under the program name PROG56.

NAME,15,TEST

If the sequence number of the next object module in the sequential library is less than 15, LIB/LIB32 copies all object modules from BI to BO until the sequence number of the next object module in the library is equal to 15. At this point, LIB/LIB32 copies this object module to logical file BO under the program name TEST.

Exit the "Verify Assembler Function Codes" Mode

The No Function (NFUNCTION) Directive places LIB/LIB32 in the mode in which LIB/LIB32 does not verify Assembler function codes. Sequence numbers and checksums are still verified if LIB/LIB32 is in the VERIFY mode. Refer to the VERIFY Directive.

While in the NFUNCTION Mode, the output from the GET Directive in the MAX IV Task/Overlay Cataloger (TOC) and from the MAX IV Link Editor (M4EDIT) can be copied and updated. In this mode, LIB/LIB32 updates standard library files much faster than in the FUNCTION Mode.

The FUNCTION Mode is the default mode when LIB/LIB32 is executed. Refer to the FUNCTION Directive.

SYNTAX

NF[UNCTION]

EXAMPLES

NFUNCTION
The NFUNCTION Mode is set.

NFU
The NFUNCTION Mode is set.

NOFILE

Exit the "Copy File-Mark Records" Mode

The NOFILE Directive places LIB/LIB32 in the mode in which the COPY Directive does not copy file-mark records onto logical file 80 every time a file-mark record is read on logical file 81.

In the NOFILE Mode, a WEOF 80 Directive normally follows a COPY Directive in order to mark the end of the new library.

The FILE Mode is the default mode when LIB/LIB32 is executed. Refer to the FILE Directive.

SYNTAX

NOF[ILE]

EXAMPLES

NOFILE
The NOFILE Mode is set.

NOF
The NOFILE Mode is set.

Do Not Enter the HOLD State Before Initializing a Directory

The NOHOLD Directive places LIB/LIB32 in the mode in which a HOLD will not be performed before initializing a directored library.

The HOLD mode is the default mode when LIB/LIB32 is executed. Refer also to the HOLD Directive.

SYNTAX

NOH[OLD]

EXAMPLES

NOH
The NOHOLD mode is set.

NOHOLD
The NOHOLD mode is set.

$NOP

Display a User Comment

The $NOP Directive can be used in a job stream or a $DO procedure to display a user's comment record.

SYNTAX

$NOP [text]

[text] - Parameter 1 (optional) consists of a user's comment composed of ASCII characters.

EXAMPLE

$NOP THIS IS MY PROCEDURE

The message $NOP THIS IS MY PROCEDURE is displayed.

Display a Message With No Response Required

The NOTE directive sends a message to the system operator's attention on the optional files LO and CO while a job stream is being executed.

Although only the first three characters (NOT) are required for the NOTE Directive, it is recommended that all four characters (NOTE) be typed to avoid confusion. (Refer to second example below.)

SYNTAX

NOT[E] [text]

[text] - Parameter 1 optionally contains the text of the message to be output to the CO file. The maximum number of characters for the text is governed by the device to which the CO file is currently assigned.

EXAMPLES

NOTE PROGRAMS ARE NOW BEING CATALOGED
The message 'NOTE PROGRAMS ARE NOW BEING CATALOGED' is output to the CO file.

NOT END OF JOB STREAM X
The message 'NOT END OF JOB STREAM X' is output to the CO file. Notice that dropping the directive's insignificant character E results in a message that is the opposite of the intended message.

NOT
The message 'NOT' is output to the CO file.

NOVERIFY

Exit the "Verify Function Codes, Sequence Numbers, and Checksums" Mode

The NOVERIFY Directive places LIB/LIB32 in the mode in which function codes, sequence numbers, and checksums are not verified. Output records are still reblocked to the device size or to the size specified in the RECORD Directive. A new checksum and sequence number is computed for output records. The NOVERIFY Mode enables LIB/LIB32 to run faster while updating object modules onto reliable devices. For example, disc has its own input checksum or its media. The NOVERIFY Directive overrides the FUNCTION Directive.

The VERIFY Mode is the default mode when LIB/LIB32 is executed. Refer to the VERIFY Directive.

SYNTAX

NOV[ERIFY]

EXAMPLES

NOVERIFY
The NOVERIFY Mode is set.

NOV
The NOVERIFY Mode is set.

Display a Message and Wait for Response

The PAUSE Directive (same as the ACTION Directive) outputs a message on the CO file directed to the operator's attention. LIB/LIB32 is placed into the HOLD state, and a /R must be entered to resume execution. To abort LIB/LIB32, enter /A.

SYNTAX

PAU[SE] [text]

[text] - Parameter 1 (optional) displays a message directed to the operator's attention. Default is the message 'PAUSE'.

EXAMPLES

PAUSE SET THE WRITE-PROTECT SWITCHES FOR DISC
The message 'PAUSE SET THE WRITE-PROTECT SWITCHES FOR DISC' is output to the CO file.

PAU SET THE WRITE-PROTECT SWITCHES FOR DISC
The message 'PAU SET THE WRITE-PROTECT SWITCHES FOR DISC' is output to the CO file.

Position a Logical File to a Specified Object Module

The POSITION directive positions a logical file to directly ahead of a specified object module. If the logical file BI is specified and has a directory, BI is positioned directly to the object module specified. In all other cases, the logical file is sequentially searched from its current position. Records that are bypassed are not verified in any way. If a file-mark record is read from the input file before the specified object module is found, LIB/LIB32 displays an error message.

SYNTAX

POS[ITION] sequence-number [logical-file]
program-name
catalog-name

sequence-number
program-name
catalog-name — Parameter 1 (required) must specify one of the following:

sequence-number specifies the sequence number of the object module. It must be a positive number. The sequence-number reflects the object module's sequential position from the beginning of logical file BI.

program-name specifies the program name of the object module. If a program-name is specified in this directive, the device must be able to backspace because LIB/LIB32 must first read the program name and then the device must be backspaced to the position ahead of the object module.

catalog-name specifies the object module's catalog name in the directory if the logical file has a directory.

[logical-file] — Parameter 2 (optional) specifies a logical file name. If a logical-file other than BI is specified, the logical file is sequentially searched from its current position. If not specified, BI is used.

EXAMPLES

POS 5	LIB/LIB32 positions logical file SI to the fifth object module in the library.
POSITION PROG34	LIB/LIB32 positions logical file SI to the object module under the program name or catalog name) PROG34.
POS ABC JL	LIB/LIB32 positions logical file JL to the object module ABC.

The POSITION Directive allows faster updating of sequential libraries. The following example illustrates this point.

$EXEC LIB,,NOLO or $EXE LIB32,,NOLO	LIB is executed with the NOLO option.
NFUN	The No Function Code Verificat on Mode is set.
ASS X ALB BO SCB POS ABCD X	Logical file ALB is to be updated. ALB is positioned to module ABCD since X is assigned to ALB.
ASS SI X	Assigns logical file SI to X, therefore SI equals ALB.
REP ABCD	Replaces ABCD. Refer to the REPLACE Directive.
COPY ASS SI SCB BO X	Copies ABCD and rest of library to SCB Assigns logical file BO to X, therefore, BO equals ALB.
COPY	Copies SCB to ALB. Object modules copied are object modules from SI as a result of the REPLACE Directive) and ABCD and the rest of the library (as a result of the COPY Directive).
EXIT	Exit from LIB.

NOTE: The first part of the library on ALB was not copied, thus saving time.

PROMPT

Enable/Disable the Command Prompt Character

The PROMPT Directive causes L B/L TB32 to generate or suppress the output of the prompt character. When enabled, the prompt character (>), is output to the CO logical file before reading the next command.

SYNTAX

PRO[MPT] [OFF]

[OFF] - Parameter 1 optional is the keyword "OFF". When specified, Paremeter 1 disables prompt character generation.

EXAMPLES

PROMPT
Turns on prompt character generation.

PRO
Enable prompt.

PRO OFF
Suppress output of the prompt character

(MAX IV only)

Set the Number of Logical Records to be Written as One Paper Tape

The Paper Tape (PTAPE) Directive (MAX IV only) causes LIB to write a logical end-of-paper-tape record ('$$B') onto the logical file BO after the specified number of logical records are written to paper tape. This directive is used only when an updated copy of a sequential library is being punched on paper tape. Large and bulky paper tapes can be divided into smaller, easier to handle tapes.

After the specified number of logical records is written to paper tape, the following message is written to the logical file CO.

REMOVE PAPER TAPE/ttt/LIB <HOLD

LIB enters the HOLD state. To continue, the operator should check the supply of paper tape in the punch, activate the Operator Communication task, and enter the /RESUME Directive.

SYNTAX

PTA[PE] integer

integer - Parameter 1 (required) specifies the number of logical records that are to be written to paper tape.

EXAMPLE

PTAPE 100

LIB enters the HOLD state and displays a message after 100 logical records are written to paper tape.

RECATALOG

Delete Entry, Copy Module and Update Directory

The RECATALOG directive causes LIB/LIB32 to:

1. Copy an object module from logical file SL to logical file UL in the largest available space.
2. Update the directory of logical file UL by entering the catalog name #FAFFFAFFFAFF as the name of the object module which was copied.
3. Delete the directory entry of catalog-name on logical file UL. This object module can no longer be accessed.
4. Rename #FAFFFAFFFAFF to catalog-name.

Although this directive has the same shortened form as the RECORD Directive, no ambiguity exists since program names must start with an alpha character and since the RECORD Directive requires a numeric parameter.

SYNTAX

REC[ATALOG] [catalog-name] [identification]

[catalog-name] - Parameter 1 (optional) specifies the catalog name to be deleted of an object module in the UL directory. The catalog-name must begin with an alpha character and not exceed eight characters in length. If not specified, LIB/LIB32 uses the program-name of the object module to be copied. If the program name is present in the UL directory as a catalog-name, it is deleted and the object module is copied to UL under the catalog-name #FAFFFAFFFAFF.

[identification] - Parameter 2 (optional) specifies an additional means of identifying the new object module. This identification will appear next to the catalog-name of the object module on a directory listing. If not specified, the date and time when the new object module is cataloged will be recorded in the directory. If specified, the identification must follow the same form as the catalog-name, that is, begin with an alpha character and not exceed eight characters in length.

EXAMPLES

RECATALOG PROG1 SMITH

LIB/LIB32 deletes the UL directory entry for object module PROG1, and copies an object module from SL to UL under the catalog-name #FAFFFAFFFAFF. The identification SMITH is also entered into the directory.

REC

If the program-name of the object module to be copied matches a catalog-name in the UL directory, the directory entry is deleted. LIB/LIB32 copies the object module from SL to UL and enters #FAFFFAFFFAFF as the catalog-name and the date and time as the identification in the UL directory.

RECORD

Specify the Record Size of Object Modules in Words per Record

The RECORD Directive specifies the record size, in words per record, of object modules. When the COPY, ADD, and REPLACE Directives are used to write binary object records to the logical file BO, the record size used is normally the system-specified record size for the device to which the logical file BO is assigned. The RECORD Directive is used to override the system-specified record size and to establish a different record size.

Although this directive has the same shortened form as the RECATALOG Directive, no ambiguity exists since a numeric parameter is required in the RECORD Directive and since a catalog-name, starting with an alpha character, is the first parameter of the RECATALOG Directive.

SYNTAX

REC[ORD] record-size

record-size - Parameter 1 required) specifies the record size, in words per record, of object modules. A decimal number must be used.

EXAMPLES

RECORD 40
The record size of object modules is set to 40 words per record.

REC 40
The record size of object modules is set to 40 words per record.

Delete the Directory Entry of an Object Module

The REMOVE Directive deletes the directory entry in the directory of the logical file JL for the object module specified. In effect, the object module under the specified catalog-name can no longer be accessed.

SYNTAX

REM[OVE] catalog-name

catalog-name - Parameter 1 (required) specifies the catalog-name under which the object module is cataloged in the directory of logical file JL. The catalog-name must begin with an alpha character and must not exceed eight characters in length.

EXAMPLES

REMOVE PROG1

The directory entry for PROG1 in the JL directory is deleted. The object module PROG1 can no longer be accessed.

REM PROG2

The directory entry for PROG2 in the JL directory is deleted, effectively removing the object module PROG2.

RENAME

Change a Catalog-Name in the Directory of the Logical File UL

The RENAME Directive causes LIBR/LIB32 to search the directory of logical file UL for the catalog-name specified in parameter 1 and to change this catalog-name to the name specified in parameter 2.

SYNTAX

REN[AME] catalog-name-1 catalog-name-2

catalog-name-1 - Parameter 1 (required) specifies the object module catalog-name to be changed. The catalog-name must be valid entry in the directory of logical file UL.

catalog-name-2 - Parameter 2 (required) specifies the new catalog-name under which the object module is to be cataloged. In both parameters, the catalog-name must begin with an alpha character and not exceed eight characters in length.

EXAMPLES

RENAME PROG1 PROG2
The object module under the catalog-name PROG1 is renamed to PROG2.

REN PROG3 PROG4
The object module under the catalog-name PROG3 is renamed to PROG4.

Copy Modules in a Sequential Library then Insert Modules from SI

The REPLACE Directive causes LIB/LIB32 to copy object modules in a sequential library from the current position of logical file BI to logical file BO. All object modules up to but not including the object module or range of object modules specified are copied. The object module to be skipped is specified by program-name or sequence number in the sequential library. A range of object modules is skipped by specifying a second program-name or sequence number.

LIB/LIB32 then proceeds to copy the object module(s) on logical file SI to logical file BO. Copying from logical file SI terminates when a file-mark record is encountered on logical file SI. LIB/LIB32 then reads the next directive from the logical file CI or AI.

NOTE: Object modules on logical file BI following the object module(s) skipped can be copied to logical file BO, replaced, deleted, or added to logical file BO by subsequent COPY, REPLACE, DELETE, or ADD Directives.

SYNTAX

REP[LACE] sequence-num-1 [sequence-num-2]
program-name-1 [program-name-2]

sequence-num-1
program-name-1 - Parameter 1 required specifies one of the following:

sequence-num-1 specifies the sequence number of the object module to be skipped. The sequence-number reflects the object module's sequential position from the beginning of the logical file BI. It must be a decimal number greater than zero.

program-name-1 specifies the program-name of the object module to be skipped. It must begin with an alpha character and not exceed eight characters in length.

[sequence-num-2]
[program-name-2] - Parameter 2 optional specifies one of the following:

sequence-num-2 specifies the sequence number of the last object module in a range of object modules to be skipped. It must be a decimal number greater than sequence-number-1.

program-name-2 specifies the program-name of the last object module in a range of object modules to be skipped. The object module specified must be subsequent to the object module specified in Parameter 1.

EXAMPLES

REPLACE ALPHA,OMEGA
LIB/LIB32 copies object modules in a sequential library from the current position of logical file BI to logical file BO up to but not including the range of object modules from ALPHA to OMEGA. LIB/LIB32 then copies object modules from logical file SI to logical file BO until a file-mark record is read on SI.

REP 3
LIB/LIB32 copies object modules in a sequential library from the current position of logical file BI to logical file BO up to but not including the third object module in the sequential library. LIB/LIB32 then copies object modules from logical file SI to logical file BO until a file-mark record is read.

REP 4 ESTER
LIB/LIB32 copies object modules in a sequential library from the current position of logical file BI to the logical file BO up to but not including the range of object modules from the fourth object module in the library to the object module under the program-name ESTER. LIB/LIB32 then copies the object modules from logical file SI to logical file BO until a file-mark record is read.

Copy a Previously Saved Directoried Library from BI to JL

RES NOD

Convert a Sequential Library from SI into a Directoried Library

RES JPD

Recatalog One or More Modules in a Directoried Library

The RESTORE Directive performs the following functions:

- Copies a previously saved directoried library of object modules from logical file BI to the logical file JL.
- Converts a sequential library from logical file SI into a directoried library on logical file JL.
- Recatalogs all modules on SI in directoried library file JL.

Copy Saved Directoried Library

When the RESTORE Directive is used without parameter 1, LIB/LIB32 copies the previously saved directory and associated object modules from logical file BI to logical file JL. The previous data on logical file JL is replaced.

CAUTION

To avoid loss of data on logical file JL, back-up logical file JL before executing the RESTORE Directive.

Convert Sequential Library to Directoried

When the RESTORE Directive is used with parameter 1 specified as NOD[IRECTORY], LIB/LIB32 converts a sequential library on logical file SI into a directoried library on logical file JL by:

1. Building the directory sectors through INITIALIZing the logical file JL.

CAUTION

To avoid loss of data on logical file JL, back-up logical file JL before executing the RESTORE DIRECTIVE.

2. Writing the directory entries and copying the object modules by performing a separate CATALOG Directive for each object module in the sequential library. The CATALOG Directive uses the program name of the object module being copied as the catalog name.

Recatalog Modules in Directoried Library

When the RESTORE Directive is used with parameter 1 specified as UPD[ATE], LIB/LIB32 performs a separate RECATALOG directive on each module on SI until it reads a file mark from SI.

Unless the NOVERIFY or NFUNCTION Mode has been specified, LIB/LIB32 verifies the sequence numbers, checksums, and function codes. LIB/LIB32 lists object module information in the current listing mode on logical file LO. Refer to the NOVERIFY, VERIFY, NFUNCTION, FUNCTION, and SAVE Directives.

SYNTAX

RES[TORE] [NOD[IRECTORY]]
[UPD[ATE]]

[NOD[IRECTORY]] - Parameter 1 optional. If not specified, LIB/LIB32 copies a previously saved directoried library from logical file BI to logical file UL.

[UPD[ATE]]

If the keyword "NOD" is specified, LIB/LIB32 converts a sequential library on logical file SI into a directoried library on logical file UL.

If the keyword "UPD" is specified, all modules on SI up to the next file mark are recataloged using the PGM name as the catalog name.

EXAMPLES

RESTORE
LIB/LIB32 copies a previously saved directory library from logical file BI to logical file UL.

RES NOD
LIB/LIB32 converts a sequential library from logical file SI into a directory library on logical file UL.

RES UPD
LIB/LIB32 updates UL by recataloging all modules on logical file SI up to the next file mark.

Rewind Device to Beginning-Of-Medium

The REW[IND] directive positions a device or devices to their beginning-of-medium (BOM) position(s). If the device(s) is not capable of being so positioned, the directive is ignored or some appropriate normalization function is performed on the device(s). The File Position Index (FPI) in the File Assign Table (FAT list) is reset to zero.

SYNTAX

REW[IND] file [file]...

file — Parameter 1 is required. (Parameters 2 through n are optional.) It specifies the logical file to be rewound. More than one logical file can be specified in the same statement.

EXAMPLES

REW B1,B0
Position to logical file B1 and B0 and rewind them.

REWIND B0
Position to logical file B0 and rewind it.

SAVE

Copy a Directoried Library from UL to BO

SAVE NOD

Convert a Directoried Library to a Sequential Library

The SAVE Directive performs the following functions:

Copies the directory and all object modules of a directoried library on logical file UL to logical file BO. BO should be assigned to a magnetic tape or disc device.

Converts a directoried library on logical file UL into a sequential library on logical file BO by copying object modules in the order of the directory and by writing a file-mark record after the last object module. The directory is not copied.

Copy Directoried Library

When the SAVE Directive is used without Parameter 1, the directoried library is copied. During the copy, the directoried library is compressed. Therefore, when the library is subsequently restored with the RESTORE Directive, the directory will appear at the beginning of logical file UL and will be immediately followed by the associated object modules. A SAVEd directoried library can only be read by the RESTORE and TEST Directives.

Convert Directoried to Sequential Library

When the SAVE Directive is used with Parameter 1, the directoried library is converted into a sequential library.

NOTE: The BO file is reassigned during both forms of the SAVE operations.

SYNTAX

SAV[E] [NOD[IRECTORY]]

[NOD[IRECTORY]] - Parameter 1 (optional). If not specified, LIB/LIB32 copies a directoried library from logical file UL to logical file BO.

If the keyword "NOD" is specified, LIB/LIB32 converts a directoried library on logical file UL into a sequential library on logical file BO.

EXAMPLES

SAVE

LIB/LIB32 copies the directory library on logical file IL to logical file BO that should be assigned to a magnetic tape or disc device.

SAV NOD

LIB/LIB32 converts a directory library on logical file IL into a sequential library on logical file BO.

SEA

Search a Sequential Library and List Matches

The SEA Directive causes LIB/LIB32 to list all occurrences of a specified name found in a sequential library. LIB/LIB32 searches the logical file BI for Internal, External and Common references that match the name specified as parameter 1. Matches are made ignoring the character positions specified as plus signs '+'.

Optional parameters 2, 3 and 4 limit the scope of the search. Any combination of Internal, Common and External definitions can be specified.

SYNTAX

SEA name [,I] [,C] [,E]

name - Parameter 1 (required) identifies the name to be matched with Common, Internal or External definitions in the library pointed to by BI. The name consists of one to six characters, each of which is either a valid character that can be used in a program name, or a plus sign. The plus sign '+' is used as a wild card character. Matches are made ignoring the character positions held by any plus signs.

[,I]
[,C]
[,E] - Parameters 2, 3 and 4 (optional) are keywords E, I and C. They can be specified in any combination in any order. The keywords limit the search to the reference type(s) specified.

I - Internal
C - Common
E - External

If parameters 2, 3 and 4 are omitted, all three types are tested in the search.

EXAMPLES

```
ASS BI AL4
SEA IS$+++
```

Search the sequential library (BI) for all references, Internal, External and Common, to names starting with the characters "IS$" and list each occurrence.

```
SEA IS$+++ E
```

Same as above, except only Externals are considered for a match.

Compress a Directoried Library

The SQUEEZE Directive compresses a directoried library by copying the directory sectors and associated object modules, starting at the beginning of the disc partition of logical file UL.

A directoried library on logical file UL needs to be SQUEEZEd when the following message appears at the end of a listing produced by the LDIR Directive:

MESSAGE:

NEEDS TO BE COMPRESSED

A SQUEEZE can contain the following four passes:

. PASS ONE	Logical file UL is copied to logical file BO. Refer to the SAVE Directive.
. PASS TWO	Logical file BO is read and data integrity checks are performed. Refer to the TEST Directive.
. PASS THREE	Logical file BO is copied back to logical file UL. Refer to the RESTORE Directive. The directory is then listed on logical file LO.
PASS FOUR	Each object module on logical file UL is read and checked for data integrity as the directory is listed. Refer to the LDIR Directive.

If the NOTEST parameter is specified, only passes one and three are executed.

Informational messages are printed during passes one and two. The following messages are output during the passes, followed by a directory listing.

MESSAGES:

SQUEEZE
SAVE COMPLETED
TEST COMPLETED

If a sequence error is encountered, the entire SQUEEZE continues. However, SEQUENCE ERROR messages are printed at each stage and in the directory listing.

SYNTAX

SQU[EEZE] [NOT[EST]]

[NOT[EST]] - Parameter 1 optionally specifies that only passes one and three are executed. In other words, no data integrity checks are made.

EXAMPLES

SQUEEZE

LIB/LIB32 compresses a directory library by executing all four passes in the SQUEEZE operation.

SQU NOT

LIB/LIB32 compresses a directory library by executing only passes one and three in the SQUEEZE operation.

Search a Directoried Library and List Matches

The SUL Directive causes LIB/L1B32 to list all occurrences of a specified name found in a directoried library. LIB/L1B32 searches the logical disc file UL for Internal, External and Common references that match the name specified as parameter 1. Matches are made ignoring the character positions specified as plus signs '+'.

Optional parameters 2, 3 and 4 limit the scope of the search. Any combination of Internal, Common and External definitions can be specified.

SYNTAX

SUL name [,I] [,C] [,E]

name - Parameter 1 (required) identifies the name to be matched with Common, Internal or External definitions in the library pointed to by UL. The name consists of one to six characters, each of which is either a valid character that can be used in a program name, or a plus sign. The plus sign '+' is used as a wild card character. Matches are made ignoring the character positions held by any plus signs.

[I]
[C]
[E] - Parameters 2, 3 and 4 (optional) are keywords I, C and E. They can be specified in any combination in any order. The keywords limit the search to the reference type(s) specified.

I - Internal
C - Common
E - External

If parameters 2, 3 and 4 are omitted, all three types are tested in the search.

EXAMPLES

SUL MY$+++
Search the directoried library (UL) for all references, Internal, External and Common, to names starting with the characters "MY$" and list each occurrence.

SUL MY$+++ E
Same as above, except only Externals are considered for a match.

ASS UL XYZ
Assign UL to a directoried library or disc.

SUL ABC I,E
Search UL for all occurrences of ABC as an Internal or External definition and list all matches.

TEST

Check the Data Integrity of the Copy

The TEST Directive checks the integrity of data resulting from the copies performed by a SAVE or SQUEEZE Directive. Records are read from the logical file BI. The checksums and sequence numbers are validated, but the functions of the binary object modules are not checked. To do this, the LIST Directive can be used.

SYNTAX

TES[T]

EXAMPLES

TEST
: LIB reads the records of object modules from the logical file BI and validates checksums and sequence numbers.

TES
: LIB reads the records of object modules from the logical file BI and validates checksums and sequence numbers.

Specify the Library Identifier of the Directoried Library on Logical File UL

The USE directive specifies a directoried library identifier to be placed in the directory of the directoried library on logical file UL. This identifier is displayed on any subsequent directory listings.

SYNTAX

USE identification

identification - Parameter 1 (required) specifies the directory library identifier. It must begin with an alpha character and not exceed eight characters in length.

EXAMPLE

USE YANK
The identifier YANK is assigned to the directoried library

VERIFY

Enter the "Verify Function Codes, Sequence Numbers, and Checksums" Mode

The VERIFY Directive places LIB/LIB32 in the mode in which function codes, sequence numbers, and checksums are verified. The VERIFY Mode is the default mode when LIB/LIB32 is executed. Refer to the NOVERIFY Directive.

SYNTAX

VER[IFY]

EXAMPLES

VERIFY
The VERIFY Mode is set.

VER
The VERIFY Mode is set.

Write an End-Of-File-Mark

The Write End-Of-File (WEOF) Directive writes an End-of-File (EOF) mark on each selected logical file.

For some devices, this is a hardware function; for others, a software equivalent is devised and detected by handlers during reading or positioning operations for standard data formats. Further information about device-handler operations can be found in the MAX IV DATA STORAGE DEVICE HANDLERS, System Guide Manual.

SYNTAX

WEOF[,] file [,file]...

file - Parameter 1 (required) specifies the name of the logical file on which the EOF is written. If not specified, the command performs no function.

[file] - Parameters 2 through n are optional and specify additional logical files on which file marks are to be written.

EXAMPLES

WEOF S1
Write one EOF on logical file S1.

WEOF S1,S0
Write EOFs on logical files S1 and S0.

Cross-Reference All Modules in a Sequential Library

The XREF Directive causes LIB/LIB32 to create a cross-reference listing of all object modules on a sequential library. The cross-reference listing includes internals, externals and common references. The listing can be limited to any combination of these references by using the optional keyword parameters with the XREF Directive.

LIB/LIB32 reads the sequential library from the logical file BI and outputs the cross-reference listing to the logical file LO. Before the list is generated, LIB/LIB32 requests a title from the user. The supplied title is printed on the top of each page of the cross-reference listing.

LIB/LIB32 allocates an 8000 word area in memory to use in ordering the cross-reference. When additional work area is required to process a lengthy cross-reference, LIB/LIB32 uses the SC and WRK scratch files.

SYNTAX

XRE[F] [E] [,I] [,C]

[E]
[I]
[C] - Parameters 1, 2 and 3 (optional) are keywords E, I and C. They can be specified in any combination in any order. The keywords limit the search to the reference types specified.

E - External
I - Internal
C - Common

If parameters 1, 2 and 3 are omitted, all three types are included in the cross-reference.

EXAMPLES

XREF
Create a cross-reference listing of all three reference types for the sequential library on BL.

XREF E,I
Create a cross-reference listing of the external and internal references for the sequential library on BL.

CHAPTER 4
GUIDE TO THE USE OF MAX IV/MAX 32 LIBRARY UPDATE

This chapter contains:

- Operating conventions of MAX IV and MAX 32 Library Update
- Examples of the use of directoried library directives
- Examples of the use of sequential library directives
- Presentations of the listing formats of LIB/LIB32
- Examples of Job Control procedures using LIB/LIB32 directives

4.1 OPERATING CONVENTIONS

The following paragraphs provide the operating conventions of LIB/LIB32.

4.1.1 Executing LIB/LIB32

Library Update is usually run as an overlay of the Job Control Batch Task. It is run by a Job Control Directive of the form:

$EXECUTE LIB,file,options

or

$EXECUTE LIB32,file,options

The file parameter specifies the logical file from which LIB/LIB32 is loaded. The options parameter includes:

- HOLD - If the HOLD option is set, LIB/LIB32 enters the HOLD state before reading any directive. The Operator Communications task must be activated and /RESUME must be entered before each directive.
- LO - If the LO option is reset, LIB/LIB32 lists only directives on the device to which the logical file LO is assigned. If set, LIB/LIB32 lists information of object modules processed.

  AO - If the AO option is set, directives are read from the device to which the logical file AI is assigned.

4.1.2 Module Names

When dealing with directoried libraries, two names are associated with an object module, the program-name and the catalog-name. The program-name is the name in the Program Name (PGM) statement within the object module. The catalog-name is the name by which the object module is cataloged in the directory. Both names may consist of from one to six Compressed Alphanumeric (CAN) code characters. The first character must be alpha. When

a program-name consists of spaces only, the default catalog-name is $BLANK. When dealing with sequential libraries, only the program-name is used.

4.1.3 Input Formats

The principal input format of records in object modules is MODCOMP's Standard Binary format. This format will be described in the DATA STRUCTURES (Volume II, System Design Manual.

If using the MAX IV processor LIB, the NFUNCTION Mode must be set to process the output from the MAX IV Link Editor (M4EDIT) and from the GET Directive in the MAX IV TASK/OVERLAY CATALOGER. Refer to the NFUNCTION Directive in this manual for more information.

If using the MAX 32 processor LIB32, output can be processed from the MAX 32 Link Editor (LNK32) directly without the use of the NFUNC Directive. To process output from the GET Directive in the MAX 32 TASK/OVERLAY CATALOGER (TOC32) the NFUNC Directive must be used.

The directoried library directives can process a sequential library if it has been converted into a directoried library through the RESTORE NODIRECTORY Directive. In addition, the directoried library directives can process a directoried library in the special SAVE format through the RESTORE Directive.

4.1.4 Directoried Library Format and Use

The directoried library directives build and maintain a directoried library accessed through a directory. The format of a directory of a directoried library is described in Appendix C. Directives that write to the directoried library take exclusive use of the library for the period of the update.

4.1.5 Output File Formats

The directoried library directives can create a sequential library through the SAVE NODIRECTORY Directive or save a directoried library in the special SAVE format through the SAVE Directive.

File Marks

Every file-mark record read from the logical file BI is written on the logical file BO unless a NOFILE Directive was previously entered.

If a NOFILE directive was previously entered and the operator wishes to restore the writing of file-mark records by subsequent COPY directives, a FILE Directive must be entered.

Record Size

LIB/LIB32 normally interrogates its binary output device (device assigned to logical file BO) and reblocks all input records into an output record size suitable for the output device. If the device has unlimited record size, a 128-word (256-byte) limit is assigned. If the device has a limited size, all records are blocked

to this limit. The RECORD Directive, when used, overrides the natural reblocking and causes a specific record size (in words) to be forced.

Paper Tape Control

The PTAPE Directive is useful when the logical file 80 is assigned to a paper-tape punch device. It puts a limit on the number of records that can be written on one paper tape.

When the specified limit is reached, a special "End-of-Roll" record ($$BI) is written on the tape, and LIB/LIB32 HOLDS after displaying this message on the logical file CO:

LIB MESSAGE

REMOVE PAPER TAPE /ttt/LIBHOLD

LIB32 MESSAGE

REMOVE PAPER TAPE /ttt/LIB32HOLD

To continue, the operator must check the supply of paper tape in the punch, activate the Operator Communication task, and enter /RESUME. The process terminates when the sequential library is completely copied.

If such consecutive paper tapes are loaded by the Task/Overlay Cataloger, a similar HOLD condition results at the end of each tape with this operator message on the logical file CO.

LIB MESSAGE

MOUNT NEXT TAPE /ttt/LIBHOLD

LIB32 MESSAGE

MOUNT NEXT TAPE /ttt/LIB32HOLD

In this manner, a large roll of tape can be split into smaller, easier to handle tapes.

4.1.6 User Control of LIB/LIB32

The PAUSE Directive is used to write a message on the logical file CO and to cause a HOLD. The PAUSE Directive is useful when LIB/LIB32 directives are entered from card decks and the operator needs to intervene at intermediate points of an updating run. The EXIT Directive transfers control to Job Control.

4.1.7 Error Handling

Unless the NOVERIFY Mode has been specified, all input record headers are verified for correct sequence numbers and binary checksums. Any out-of-sequence or improperly checksummed record causes LIB/LIB32 to write an operator error message, for example:

CHECKSUM ERROR

SEQUENCE ERROR

The full list of error and warning messages is given in Appendix A.

If the logical file CI is assigned to a device other than a terminal device, LIB/LIB32 enters a HOLD when an error message is printed. LIB/LIB32 can be continued (resumed) or aborted by the appropriate operator communication task directive (/R or /A).

If the program is resumed or if the logical file CI is assigned to a terminal device, processing continues in the following way:

- The SAVE, RESTORE, TEST, and SQUEEZE Directives can continue from the point of interruption. The output from these operations is not valid at the point of error.
- If an error occurred during the processing of any other directive, there is no recovery and LIB/LIB32 reads the next directive.

NOTE: When the processing of a directive is abandoned, file pointers can be left in undefined positions.

4.2 DIRECTORIED LIBRARIES

LIB/LIB32 is available with directoried library support. Directoried Libraries offer a convenient and efficient method of updating object modules. When the average size of object modules in a library is more than two sectors, scanning of a library (Example: By the Link Editors) (LIB works with M4EDIT and LIB32 works with LNK32.) is generally more efficient if the library has a directory.

The following sub-sections describe the uses of the directoried library directives, with additional uses of some sequential library directives when used with a directoried library.

4.2.1 Logical Files

The following logical files are used for directoried library operations.

LOGICAL FILE	USE
BI	A directoried library that has previously been saved can be RESTOREd from the logical file BI. A module can be listed or extracted from a directoried library on logical file BI.
BO	A directoried library can be saved on the logical file BO. Object modules can be extracted from a directoried library onto logical file BO. A sequential library can be created on logical file BO.
SI	Object modules to be added to or to replace existing object modules in a directoried library are read from logical file SI. A sequential library can be read from logical file SI.
UL	Directoried libraries are maintained on the logical file UL.

LO	Listings of directories, directoried libraries, and object module attributes are listed on logical file LO. Entered directives are listed on the logical file LO. Information on processed object modules is listed on logical file LO.
CO	Error messages are written on logical file CO.
CI	If the AO option is reset, directives are read from logical file CI.
AI	Directives are read from logical file AI when the AO option is set.

4.2.2 Initializing a Directory

When a new directoried library is to be created, the INITIALIZE Directive is used. An empty directory structure is built on the file assigned to logical file UL.

4.2.3 Adding Modules to the Library

Modules are added to a directoried library from logical file SI. If a new entry in the directory is to be created for an object module, the CATALOG Directive is used. If an existing object module is to be replaced by a new object module, RECATALOG is used.

EXAMPLE

Logical file SI contains object modules ALPHA and BETA in that order. The directoried library on logical file UL already contains object modules under the same names. The directoried library must be updated in the following ways:

1. Replace the object module ALPHA on UL with the object module ALPHA on SI.
2. Retain the original object module BETA on UL.
3. Catalog the object module BETA on SI under the catalog-name NEWBETA.

The following directives accomplish this workflow.

REWIND SI
: Logical file SI is rewound.

RECATALOG
: The directory entry for object module ALPHA on UL is deleted. The object module ALPHA on SI is copied to UL under the catalog-name ALPHA. Refer to the RECATALOG Directive.

CATALOG NEWBETA,WASBETA
: The object module BETA on SI is copied to UL under the catalog-name NEWBETA. The entry WASBETA is entered into the identification field of the directory. Refer to the CATALOG Directive.

4.2.4 Maintaining Directoried Libraries

L st ngs of the directory entries are produced by the LDIR Directive. An ident fcat on can be given to the directoried library by the USE D rective.

Object modu.es are renamed or de eted from the director ed .ibrary by the RENAME and REMOVE D rectives.

When a brary is fragmented, the message NEEDS TO BE COMPRESSED is pr nted on the directory .st.ng. The SQUEEZE D rective is used to compress a director ed l brary.

Directoried librar.es can be saved as sequent al libraries or as directoried l brar.es on magnetic media by the SAVE Direct ve. A SAVEd directoried l brary can be val dated and restored by the TEST and RESTORE Direct.ves respect.ve.y.

4.2.5 Converting Library Formats

A sequent al .brery or .ogical file SI can be converted into a director ed library on logical file UL through the RESTORE NODIRECTORY Directive.

EXAMPLE

A compile has .ust produced three object modules on the logica f le BO: MAIN, SUB1, and SUB2. These object modules are effectively a sequentia. .brary. A directoried .brary can be created from them through the fo.ow.ng workflow.

$WEOF BO
Writes a f le-mark record on logica. fi.e BO.

$ASS SI=BO
Assigns logical fi.e SI to .ogical f le BO.

$REW SI
Rew.nds .ogical file SI, in effect, BO.

$EXE L IB
or
$EXE LIB32
Executes Library Update.

NOVERIFY
Sets the NOVERIFY Mode in LIB/LIB32.

RESTORE NODIRECTORY Init ates the RESTORE NODIRECTORY Directive.

INITIALIZATION REQUESTED <HOLD>

This is an informationa. message displayed by L IB/LIB32. LIB/LIB32 displays th s message because the f rst action taken by L IB/LIB32 s the init.a. zation that s, the bu.ding of d rectory sectors) of logica. f .e UL. Use this direct ve with caution to avoid loss of data f les on UL.

/R

Resumes LIB/LIB32. LIB/LIB32 builds directory sectors on logical file UL and begins to copy each object module on logical file BO (since SI is assigned to BO) to logical file UL.

CAT

This is an informational message displayed by LIB/LIB32. LIB/LIB32 displays this message to inform the user that the first object module, MAIN, is copied and an entry is made in the directory.

CAT

LIB/LIB32 displays this informational message for the second object module copied, SUB1.

CAT

LIB/LIB32 displays this informational message for the third object module copied, SUB2.

A directoried library on logical file UL can be converted into a sequential library on logical file BO through the SAVE NODIRECTORY Directive.

EXAMPLE

```
$EXE LIB
or
$EXE LIB32
```

Executes Library Update.

```
REW BO
```

Rewinds logical file BO.

```
SAVE NODIRECTORY
```

Copies object modules in the order of the directory and writes a file-mark record after the last object module. The directory is not copied.

```
EXIT
```

Terminates Library Update and returns to Job Control.

4.2.6 Extracting and Listing Object Modules

An object module can be extracted from a directoried library on logical file UL to logical file BO through the following workflow.

EXAMPLE

```
$EXE LIB
or
$EXE LIB32
```

Executes Library Update.

ASSIGN BI=JL
Assigns logical file BI to logical file JL.

POSITION PROG1
Positions logical file BI (that is, JL since BI is assigned to JL) to the object module under the catalog-name PROG1.

LNAME
Sets the LNAME Mode in LIB/LIB32. In this mode, only the program or catalog name is listed on logical file LO when certain directives such as COPY are initiated.

COPY
Copies the object module PROG1 to logical file BO.

EXIT
Terminates Library Update and returns to Job Control.

The attributes of a single object module can be listed from a directory library on logical file JL to logical file LO through the following workflow.

EXAMPLE

$EXE LIB
or
$EXE LIB32
Executes Library Update.

ASSIGN BI=JL
Assigns logical file BI to logical file JL.

POSITION PROG1
Positions logical file BI (that is, JL since BI is assigned to JL) to the object module under the catalog-name PROG1.

LIST
The attributes for object module PROG1 is listed on logical file LO.

EXIT
Terminates Library Update and returns to Job Control.

All the attributes of object modules with a directory library on logical file JL can be listed on logical file LO through the following workflow.

EXAMPLE

```
$RUN LIB
or
$RUN LIB32
```
Executes Library Update.

```
ASSIGN BI=JL
```
Assigns logical file BI to logical file JL.

```
REWIND BI
```
Rewinds logical file BI (that is, JL since BI is assigned to JL).

```
LIST
```
The attributes of all object modules are listed on LO since JL is POSITIONed to the beginning of the directory library as a result of the REWIND directive.

```
EXIT
```
Terminates Library Update and returns to Job Control.

4.3 SEQUENTIAL LIBRARIES

Sequential libraries were the only form of libraries supported by the earlier revisions of Library Update. This form of object module library is necessary on configurations without disc storage.

The following sub-sections describe the uses of the sequential library directives.

4.3.1 Logical Files

LOGICAL FILE USE

The following logical files are used for sequential library operations.

BI	The object modules in the sequential library to be updated are read from the logical file BI.
BO	The object modules that are included in the updated sequential library are written on the logical file BO.
SI	Object modules to be added to a sequential library or to replace existing modules in a sequential library are read from the logical file SI.
LO	A listing of the entered directives and of the sequential library (that is, names of internals, externals, common blocks, modules, and module sizes in words) are listed on the logical file LO.
CO	Error messages are written on logical file CO.
CI	If the AO option is reset, the control directives are read from the logical file CI.
AI	The control directives are read from logical file AI when the AO option is set.

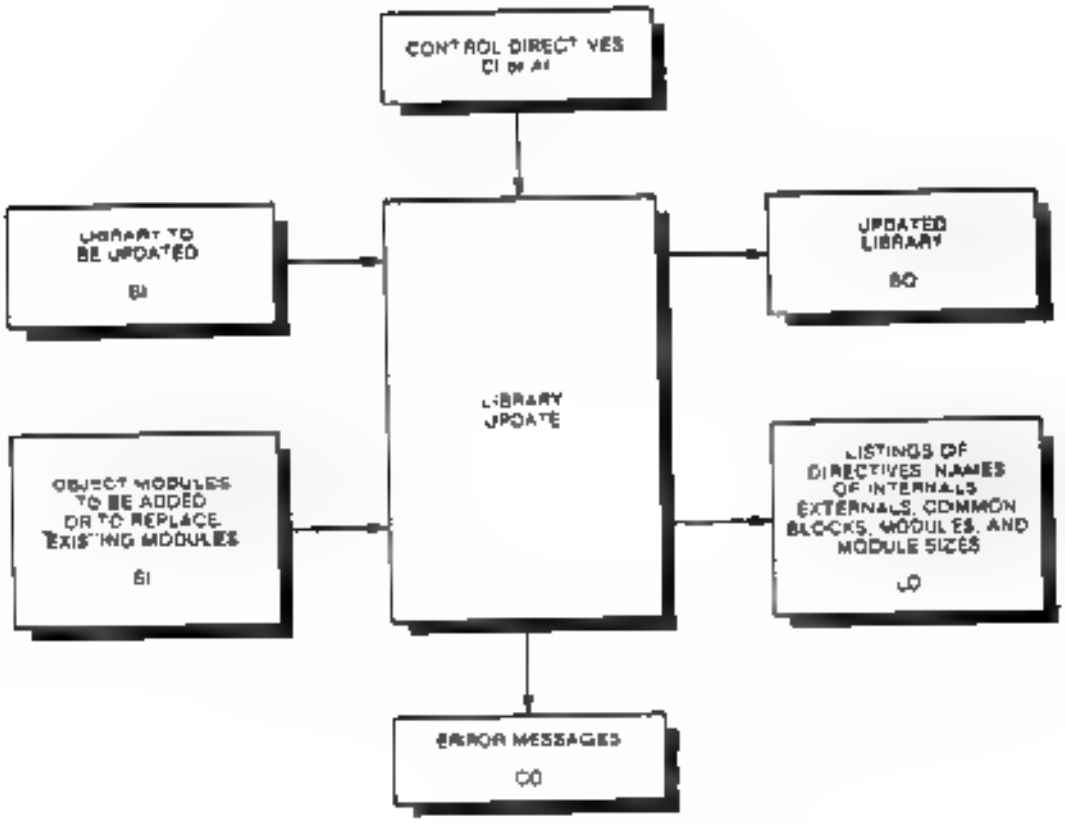


Figure 4-1. Use of Logical Files by LIB/LIB32

4.3.2 Selective Editing of Libraries by Sequence Number

Several directives are used to create, to add object modules to, and to delete object modules from a sequential library. The original copy of a library is read from logical file BI. The index used to control the update of a library is a sequence number (n) that starts with the value 1 for the first object module encountered on logical file BI. If the number ng of object modules in the original library is unknown, the user should first obtain a listing of the original library. Refer to the LIST D rective. The sequence numbers of the original library must be used to specify edit ng functions. As the updated library is produced on the logical file BO, a new listing (if enabled) specifies a new number ng of object modules.

A additions to the original library must be read from a third logical file called SI. Its name (Source Input) is not significant since it is used to read object (binary) information. This file can be assigned to any appropriate input device or logical file.

If logical file SI is assigned to a paper-tape reader the message "MOUNT ADDITIONS ON PR:ttt/ <HOLD" is written on the logical file CO before the addition takes place. If both logical files BI and SI are assigned to the paper-tape reader, a series of messages is written to the operator to allow changing and positioning of media through Operator Communications Directives:

```
MOUNT ADDITIONS ON PR/ttt/ <HOLD
MOUNT LIBRARY ON PR/ttt/ <HOLD
```

4.3.2.1 Adding Object Modules To Original Library File

If a library of new objects modules resides on logical file SI, the ADD Directive can be used to merge these object modules with an original library of object modules on logical file BI.

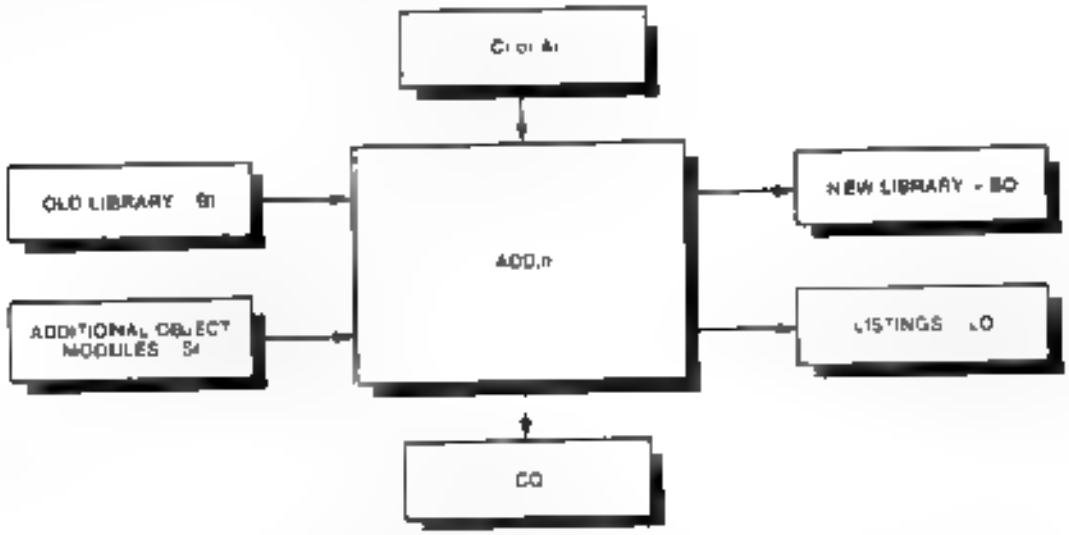


Figure 4-2. Use of Logical Files by the ADD Directive

The original library is copied from logical file BI to logical file BO up through the object module specified by sequence-number n. The additional object modules are then copied from logical file SI to logical file BO until a file-mark record is detected on logical file SI. No file-mark record is written to logical file BO. LIB/LIB32 then waits for the next directive.

If additional object modules are to be copied before the first object module in the original library, the value of n must be zero. Any number of sequential libraries of additional object modules can be copied before the first original object module by repeated use of the directive. When additional object modules are copied before the first original object module, LIB/LIB32 does not perform the copy of logical file BI to logical file BO. In other words, additional object modules are copied before the first original object module but the original sequential library is not copied. The COPY Directive must follow the last ADD Directive to copy the original sequential library after the additional object modules. If the FILE Mode is set, the COPY Directive also copies the file-mark record at the end of the original sequential library.

If the first part of the sequential library is not to be copied, the POSITION Directive can be used to reduce processing. Refer to the POSITION Directive.

4.3.2.2 Deleting Object Modules From Original Library File

If one or more contiguous object modules in the original library file are to be deleted, the DELETE Directive is used.

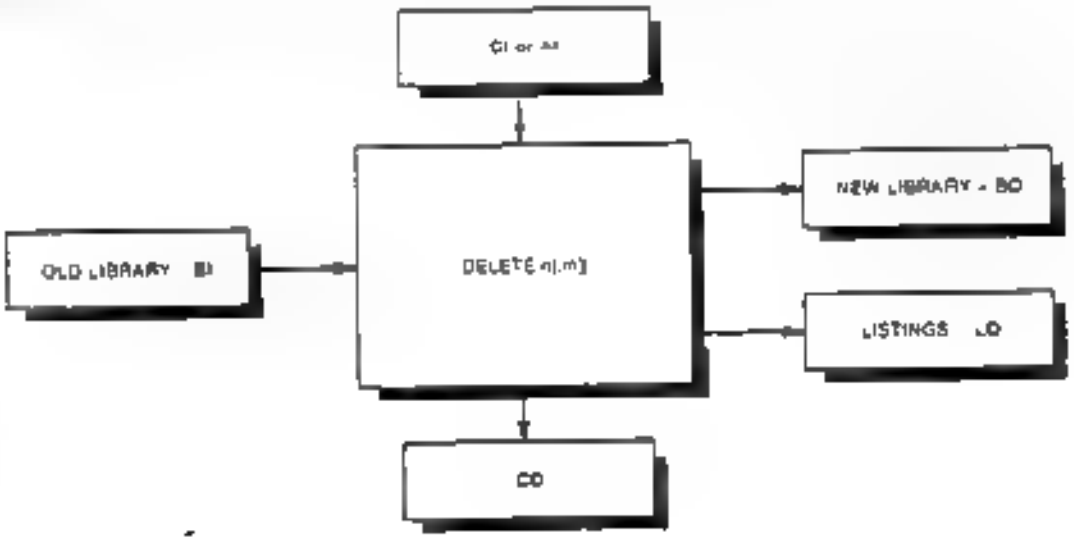


Figure 4-3. Use of Logical Files by the DELETE Directive

If the object module specified by n has not been encountered, all object modules **before** n are copied from logical file SI to logical file SO. Object module n is bypassed and, if a second parameter m is specified, additional object modules are bypassed up **through** sequence number m. LIB/LIB32 then waits until the next directive is entered. If logical file SI is positioned immediately in front of object module n, no output to logical files SO or LO results. If this is the last editing function, the COPY Directive must follow to copy the remainder of the original library to logical file SO.

4.3.2.3 Replacing Object Modules in Original Library

The REPLACE Directive is a convenient combination of the ADD and DELETE Directives. If a library of new object modules resides on logical file SI and old copies of these object modules are embedded within the original copy on logical file SI, the old object modules can be deleted and the new object modules (and any other additional object modules) can be added in a single operation.

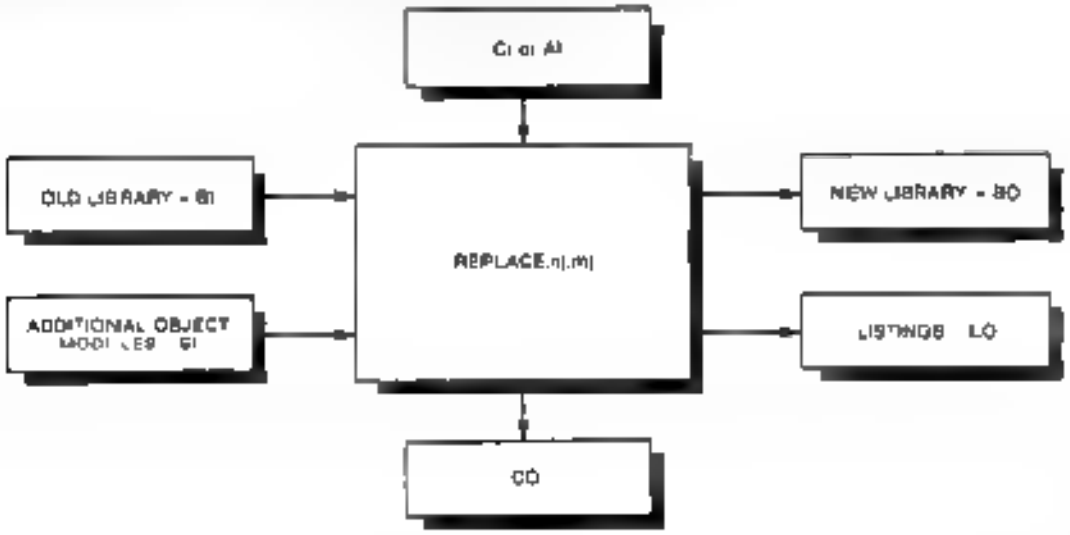


Figure 4-4. Use of Logical Files by the REPLACE Directive

If object module n has not yet been encountered, all object modules **before** n are copied from logical file SI to logical file SO. Object module n or object modules n **through** m are bypassed. All additional object modules are copied from logical file SI to logical file SO until a file-mark record is detected on logical file SI. LIB/LIB32 then waits for the next directive. If this is the last editing function, the COPY Directive must follow to copy the remainder of the original library (and its file-mark record) to logical file SO.

4.3.2.4 Sequential Referencing of Object Modules

A check for the sequential referencing of object modules is performed by the ADD, DELETE, and REPLACE Directives. The error message "STATEMENT ERROR" is written on the logical file CO when the referencing of the object module is not sequential.

After writing the error message, LIB/LIB32 proceeds to read the next directive.

Consider, for example, the following directives:

```
ADD 1
REP 7,5
```

The second directive REP 7,5 cannot be processed by LIB/LIB32 because object module 5 precedes object module 7 in the library. In other words, a range of 7 through 5 is invalid.

4.3.2.5 Reset of Program Sequence Number

The input index used to control most editing functions is the sequence number (n). This number is reset under the following conditions:

When LIB/LIB32 is initially loaded.

- After each COPY or LIST Directive has finished.
- After each ASSIGN or REWIND Directive that references the input logical file BI has been executed.

The output index which appears on the listing of the updated library is reset under the following conditions:

When LIB/LIB32 is initially loaded.

- After each COPY or LIST Directive if the NOFILE Mode has not been set.
- After each ASSIGN, REWIND, or WEOF Directive that references the output logical file BO has been executed.

4.3.2.6 Single Program Fetch

The GET,n Directive causes the nth object module to be copied from logical file BI to logical file BO.

If BI is positioned before object module n, all object modules are bypassed until n is found. Object module n is then copied to logical file BO. This directive is used to retrieve copies of a single object module from a library. LIB/LIB32 then proceeds to read the next directive. A WEOF BO Directive can be entered at this point to have a file-mark record written at the end of the retrieved object module.

4.3.3 Selective Editing of Libraries by Program Name

Only sequence numbers were used as edit references in the above editing directives. The program name of each object module can be substituted for any or all sequence number arguments in the directives listed below:

Edit by Number	Edit by Name
ADD n	ADD program-name
DEL[ETE] n	DEL[ETE] program-name
DEL[ETE] n,m	DEL[ETE] program-name1, program-name2
REP[LACE] n	REP[LACE] program-name
REP[LACE] n,m	REP[LACE] program-name1, program-name2
GET n	GET program-name

Sequence numbers n or m and program-name parameters can be intermixed, if desired. When using program-names however, the user should take care to ensure that all elements of the library have unique names.

4.3.4 Renaming an Object Module in Original Library

The NAME,n,program-name Directive causes a library to be read from logical file BI and written to logical file BO with the original program-name embedded in the nth object module changed to the specified program-name.

4.3.5 Examples of Sequential Directives

The following example illustrates a job stream that creates a library consisting of object modules that are contained in three separate libraries. The three libraries are read sequentially from the logical file BI and the new library is written on the direction to which the logical file SO is assigned. After the library is created, the program names and sizes of the object modules are listed on the listing device.

The following job stream can be read from the card reader to accomplish the desired objectives:

EXAMPLE

$JOB
Identifies a job stream.

$ASS BI=BI BO=SO LO=NO
Assigns logical files BI to BI, BO to SO, and LO to the Null device (NO) to inhibit printing.

$EXE LIB
or
$EXE LIB32
Executes Library Update.

NOFILE
Sets the NOFILE Mode in which file-mark records are NOT written to BO.

COPY 3
Initiates the COPY Directive. LIB/LIB32 copies all object modules from logical file BI to logical file SO (because BO is assigned to SO) until three file-mark records are read on BI.

WEOF BO
Writes a file-mark record on logical file SO because BO is assigned to SO.

ASS BI=SO LO=LO
Assigns logical files BI to SO and LO to LO to permit printing.

LNS
Sets the LNS Mode in which program name and size of object modules are listed.

LIST
Lists the sequential library on logical file SO because BI is assigned to SO.

EXIT
Terminates Library Update and returns to Job Control.

The next example illustrates a job stream that updates a library. The original library is read from logical file BI and the updated library is written on logical file BO. The replacement object modules reside in a pre-arranged sequential order on the disc partition to which logical file SI is assigned.

The original library is updated in the following manner:

1. Object modules 2 through 6 are deleted from the original library.
2. A separate library on logical file SI is inserted immediately after the object module under program name TEST12.
3. Object module 15 is replaced by the object modules in the next library on logical file SI.
4. A listing of program names, internals, externals, et cetera, is printed on the system listing device.

The following job stream can be read from a card reader to accomplish the described objectives.

EXAMPLE

$JOB Identifies a job stream.

$ASS BI=BI BO=BO SI=SI LO=LO
Assigns logical files BI to BI, BO to BO, SI to SI, and LO to LO.

$EXE LIB
or
$EXE LIB32
Executes Library Update.

DEL 2,6
Deletes object modules 2 through 6 from the original library; in other words, these object modules are not copied to the updated library on logical file BO.

ADD TEST,2
LIB/LIB32 continues to copy object modules from BI to BO through the object module under program name TEST12. LIB/LIB32 then copies object modules from SI to BO until the first file-mark record is read on SI.

REP ,5
LIB/LIB32 continues to copy object modules from the current position of BI to BO up to but NOT including the object module with sequence number 15. LIB/LIB32 skips this object module and then copies object modules from SI to BO until a file-mark record is read on SI.

COPY
LIB/LIB32 continues to copy object modules from the current position of BI to BO until the file-mark record of the original library is read on BI.

EXIT
Terminate LIB/LIB32 and return to Job Control.

NOTE: A listing is produced on the system listing device even though the LIST Directive is not present in the job stream. The DEL, ADD, REPLACE, and COPY Directives automatically produce listings as part of their function.

The next example illustrates a job stream that extracts a copy of an object module in a library. The library is read from logical file LB and the object module under program-name TIME is written on logical file PO in 40-word records.

The following job stream can be read from the card reader to accomplish the described object ve.

EXAMPLE

$JOB
Identifies a job stream.

$ASS BI=LB
$ASS BO=PO
$ASS LO=LO
Assigns the logical files BI to LB, BO to PO, and LO to LO.

$EXE LIB
or
$EXE LIB32
Executes Library Update.

REC 40
Sets the record size of output object modules to 40 words.

GET TIML
LIB/LIB32 reads the specified object module from the sequential library on logical file LB (because BI is assigned to LB) and writes the object module to logical file PO (because BO is assigned to PO).

WEOF BO
Writes a file-mark record on logical file PO because BO is assigned to PO.

EXIT
Terminate LIB/LIB32 and return to Job Control.

4.4 LISTING FORMATS

The listing formats of directory listings and object module listings follow.

4.4.1 Directory Listing Format

A listing of the directory of a directoried library is produced by the LDIR Directive on logical file LO in the following form:

```
DIRECTORY LISTING OF fname I.D.=id  LAST MODIFIED mm/dd/yy

MODULE    IDENTIFICATION      FILE    START    END     LENGTH

mname     id=m                 UL       1       n        n
  .         .
  .         .
  .         .
mname-2   mm/dd/yy hh:mm       UL       x       y        z
  .         .
  .         .
  .         .                   ...     ...     ...      ...

NUMBER OF ENTRIES CATALOGED = n
NUMBER OF DIRECTORY ENTRIES USED = n
NUMBER OF UNUSED DIRECTORY ENTRIES = n
LARGEST NUMBER OF CONTIGUOUS UNUSED SECTORS = n
TOTAL NUMBER OF UNUSED SECTORS = n
TOTAL NUMBER OF SECTORS USED = n
NEEDS TO BE COMPRESSED
```

Figure 4-5. Directory Listing Format

The last seven lines of information in the format are printed when logical file UL is assigned to a disc. This seven lines of directory summary information can be listed by the LDS Directive.

The NUMBER OF DIRECTORY ENTRIES USED includes an entry for each object module, a beginning directory entry, an end directory entry, and an entry for each directory sector. The NUMBER OF UNUSED DIRECTORY ENTRIES is the number available before another directory sector is added. The LARGEST NUMBER OF CONTIGUOUS UNUSED SECTORS is the size of the largest object module that can be cataloged. The NEEDS TO BE COMPRESSED message means that there are unused sectors scattered throughout the library. If a SQUEEZE is executed, these unused sectors are placed at the end of the library and the number of contiguous unused sectors will increase. If the logical file is assigned to a File Manager file, fname is FM.

4.4.2 Object Module Listing Formats

Object module information is listed on the logical file LO when object modules are copied from logical file BI to logical file BO by a sequential library directive or when object modules are listed from logical file SI. The differences in a LIB32 module listing will be shown in parenthesis throughout this section. This information consists of:

- Program name
- Common block definition name and size
- Internal definition name, counter number, and hexadecimal address

  Extended common block definition name and size (MAX 32 Operating System does not have extended common, therefore LIB32 will not show this information)

- External definition name

  Attributes assignment counter number and hexadecimal address

Listing of this information can be suppressed by specifying the NOLO option. The amount of information can be reduced by specifying the NFUNCTION or NOVERIFY Mode or the LNAME or LNS Mode.

When a directoried library is specified, the listing of object module information is produced in the following form:

```
MODULE LISTING OF fname

mname-1 id=m
    module information
mname-2 mm/dd/yy hh:mm
    module information
```

Figure 4-6. Format of Object Module Information for a Directoried Library

The mname-1 and mname-2 lines contain the catalog-name and identification information from the directory.

When a sequential library is listed or copied or an individual object module is copied or cataloged, the listing is produced in the following form:

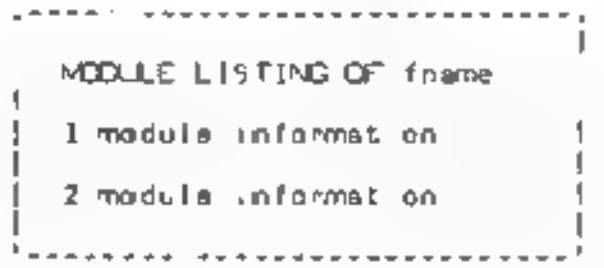


Figure 4-7. Format of Object Module Information for a Sequential Library or Individual Module

The MODULE LISTING line appears only when the listing is produced by the LIST Directive.

The module information line takes the same form in both formats and is described below. The program name, common block definitions, internal definitions, external definitions, and attribute assignments are listed in the following format:

COLUMN	CONTENTS
1-3	The sequence number of the object module within the library. This number appears only on the first line of information for each program. Spaces appear in these columns for all other lines.
4-5	Spaces
6	P for a program name C for a common block definition I for an internal definition X for an extended common block definition E for an external definition A for an attribute assignment
7	Space
8-13	If column 6 contains a P, C, I, X, or E, the name of the program, of the common block definition, of the internal definition, of the extended common block definition, or of the external definition, respectively, is printed in this field. If column 6 contains an A, a 4-digit hexadecimal attribute, followed by two spaces, is printed in this field.
14	Space

15-21	If a column 6 contains a P or E, this field contains spaces. If column 6 contains a C, this field contains a space, the 4-digit hexadecimal size (or the 8-digit hexadecimal size if LIB32) associated with the common block definition, and two spaces. If column 6 contains an X, this field contains a space and the 6-digit hexadecimal size associated with the extended common block definition. (There are no extended common blocks in MAX 32.) If column 6 contains an I or A, this field contains a 2-digit hexadecimal counter number, a space, and a 4-digit hexadecimal address (or a 8-digit hexadecimal address if LIB32) associated with an internal definition or an attribute assignment.
22-27	Spaces
28-49	Same format as columns 6-27
50-71	Same format as columns 6-27
72-93	Same format as columns 6-27

LIB/LIB32 examines the size of the records on the object file to and uses as many of the sets of columns as possible: Example 28-49, 50-71, and 72-93. Information about the counters used within the object module is listed in the following format:

COLUMN	CONTENTS
1-4	Spaces
5-13	"COUNTER"
14	Space
15-16	Hexadecimal number of the counter
17-21	Spaces
22-25	"LAST"
26	Space
27-30	Hexadecimal address last represented by the counter
31-35	Spaces
36-43	"BOUNDARY"
44	Space
45-48	Hexadecimal boundary value for counter

49-53	Spaces
54-57	"SIZE"
58	Space
59-62	Highest value ever assigned to counter in hexadecimal
63	Space
64-68	Highest value ever assigned to counter in decimal

The following information about the object module is also listed:

COLUMN	CONTENTS
1-4	Spaces
5-22	"TRANSFER ADDRESS"
23	Space
24-30	2-digit hexadecimal counter number, space, 4-digit hexadecimal transfer address (or 8-digit hexadecimal transfer address if -B32)
31-35	Spaces
36-41	"COMMON"
42	Space
43-46	Total size of all common blocks in hexadecimal
47	Space
48-52	Total size of all common blocks in decimal

For programs which do not reference counters, the counter zero is used to refer to absolute addressing and the counter one is used to refer to relocatable addressing in the above descriptions.

nformation about the use of extended common blocks s listed as follows:

COLUMN	CONTENTS
1-6	Spaces
7-21	"EXTENDED COMMON"
22	Space
23-28	Tota, size of all extended common blocks in hexadecimal
29-31	Spaces
32-38	Tota, size of all extended common blocks n decimal

```
*H_0 MODCOMP LIBRARY UPDATE    DATE 09/05/84   14:23:42   PAGE 1

    1 P UVWXYZ                C  COMN    0064      I TEMP1   08 0000
      E ABCDEF                A  0000   06 0000    X QRSTUV  08 8000
      COUNTER 06        LAST 0002      BOUNDARY 0001   SIZE 0002 2
      COUNTER 0A        LAST 00AD      BOUNDARY 0080   SIZE 00AD 173
      TRANSFER ADDRESS   0A 00AD       COMMON 0064 100
      EXTENDED COMMON     008000    32768

 TOTAL RECORDS WRITTEN =      2
```

Figure 4-8. Object Module Information Listing from LIB

```
 *A 0 MODCOMP 32-BIT LIBRARY UPDATE   02/11/85   11:43.20   PAGE 1
 MODULE LISTING OF SFMS
    1 P UVWXYZ      C  COMN    00000064      I TEMP1   08 00000000
      E ABCDEF      A  0006   06 00000000
      COUNTER 06   LAST 0000000D2   BOUNDARY 0001   SIZE 00000002 2
      COUNTER 0A   LAST 0000000AD   BOUNDARY 0001   SIZE 000000AD 173
      TRANSFER ADDRESS   0A 00000009      COMMON 00000064 100

 TOTAL RECORDS WRITTEN =      2
```

Figure 4-9. Object Module Information Listing from L1832

4.5 JOB CONTROL PROCEDURE EXAMPLES

The following Job Control procedures can be used to perform sequential library update functions. If using the procedures for 16-bit libraries you can use all the procedures as shown. If using the procedures for 32-bit libraries modify the following command:

$EXE LIB

to

$EXE LIB32

4.5.1 LOCLIB Procedure

The procedure LOCLIB lists, creates and copies object module libraries. It creates a single sequential library from several libraries, copies one or more libraries from one medium to another, or lists one or more libraries.

Parameter 1 (required) must be LIST, CREATE, or COPY.

- Parameter 2 (required) must be the input device or logical file from which object modules are to be read.
- Parameter 3 (required for a CREATE or COPY) must be the device or log call file onto which object modules are to be written. If the first parameter is LIST, this parameter is not entered.
- Parameter 4 (optional) is the number of libraries that are to be read. If this parameter is not entered, the default value is 1.

Parameter 5 (optional) is either /ALL or /NS. If /ALL is specified, program names, internals, externals, et cetera are listed. If /NS is specified, only the program names and sizes are listed. If this parameter is not entered, the default value is /NAM (that is, only program names are listed).

SYNTAX

The $DO command syntax for the CREATE function in this procedure is:

```
$DO  LOCLIB,CREATE,input-file,output-file,number-of-files[,/ALL]
                                                         [,/NS ]
```

The $DO command syntax for the COPY function in this procedure is:

```
$DO LOCLIB,COPY,input-file,output-file[,number-of-files][,/ALL]
                                                        [,/NS ]
```

The syntax of the $DO command for the LIST function in this procedure is:

```
$DO LOCLIB,LIST,input-file[,,number-of-files][,/ALL]
                                             [,/NS ]
```

EXAMPLES

$DO LCCLIB,CREATE,BI,BO,3,LALL
One sequential library is created on logical file BO from three sequential libraries on logical file BI. All object module information is listed on logical file LO.

$DO LCCLIB,COPY,BI,BO,4,LNS
Four sequential libraries on logical file BI are copied as four sequential libraries to logical file BO. Only the program name and size are listed on logical file LO.

$DO LCCLIB,LIST,BI,,2,LALL
Object module information for the object modules in two sequential libraries on logical file BI is listed on logical file LO. All object module information is listed.

Note that two commas are needed between the input file parameter (BI) and the number of files parameter (2) to skip the unnecessary output file parameter.

A listing of the procedure follows:

```
$PROC LCCLIB,,,,1,,NAM
$IFP %3,P=$ASS B:%2 BO=%3 LO=LO;$ASS BI=%2 LO=LO
$IFP %3,P=$REW BI BO;$REW BI
$EXE LIB
%5
$IF %1=CREATE,P=AVR CI=3
%1 %4
EXIT
$AVR CI=8
NOFILE
COPY
EXIT
$COUNT %4,3
$EXE LIB
%5
BKR CI=7
$WEOF BO
$NOTE END OF LCCLIB %1
```

4.5.2 UPLIB Procedure

This procedure updates an existing sequential library in one of the following ways:

1. Adding to the library up to five different libraries. Each library is inserted following the specified object module(s). The object module(s) is specified by a sequence-number(s) (n1,n2,...).

2. Replacing up to five object modules, specified by a sequence-number, (n1,n2,n3,...) with a corresponding number of libraries.

3. Deleting from the library up to five object modules specified by a sequence-number (n1,n2,n3,...).

An explanation of each parameter follows:

- Parameter 1 (required) specifies ADD, REPLACE, or DELETE.

- Parameter 2 (required) specifies the device name or logical file from which the original library is to be read.

- Parameter 3 (optional) specifies the device name or logical file on which the updated version of the library is to be written. If this parameter is not entered, the updated library is written onto the system's main scratch logical file (SC) and then copied back onto its original medium.

  Parameter 4 (required for an ADD or REPLACE) specifies the device name or logical file from which addition/replacement libraries are to be read.

  Parameters 5 through 9 (n1,n2,n3,n4,n5), are object module sequence-numbers determined from the original library listing. Parameter 5 (n1) is required. Parameters 6 through 9 are optional.

SYNTAX

The syntax of the $DO command for the ADD function in this procedure is:

$DO JPLIB,ADD,input-file,[output-file],additions-file,n1[,n2][,n3][,n4][,n5]

The syntax of the $DO command for the REPLACE function in this procedure is:

$DO JPLIB,REPLACE,input-file,[output-file],replacements-file,n1[,n2][,n3][,n4][,n5]

The syntax of the $DO command for the DELETE function in this procedure is:

$DO JPLIB,DELETE,input-file,[output-file],,n1[,n2][,n3][,n4][,n5]

EXAMPLES

$DO JPLIB,ADD,BI,BO,WRK,3,6,9

Three sequential libraries, located on logical file WRK, are added to the sequential library on logical file BI. The three libraries are added after the third, sixth, and ninth object modules in the original library. The updated library is written to logical file BO. Only the program names of all object modules in the updated library are listed on logical file LO.

$DO JPLIB,REPLACE,BI,,WRK,3,5

Two sequential libraries, located on logical file WRK, replace the third and fifth object modules in the original library on logical file BI. The updated library is written to logical file SC and then copied back to BI.

$DO JPLIB,DELETE,BI,BO,,7

The object module with sequence-number 7 is deleted from the sequential library on logical file BI. The updated library is written to logical file BO.

Note that two commas are needed between the output file parameter (BO) and the sequence-number parameter 7 to skip the unnecessary additions/replacements parameter.

A listing of the procedure follows

```
$PROCEDURE JPLIB
$ASS BI=%2 LO=LO
$IFM %3,P=$ASS BO=SC;$ASS BO=%3
$IFP %4,P=$ASS SI=%4
$REW BI BO
$EXE LIB
LNAM
$IFP %5,P=%1 %5
$IFP %6,P=%1 %6
$IFP %7,P=%1 %7
$IFP %8,P=%1 %8
$IFP %9,P=1% %9
COPY
REW BI BO
ASS BI=SC BO=%2
$IFM %3,P=COPY
REW BI BO
EXIT
$NOTE END OF JPLIB %1
```

4.5.3 GET Procedure

This procedure copies an object module from a sequential library.

- Parameter 1 (required) specifies either the object module sequence number, determined from a previous library listing, or the 1 to 6-character program-name of the object module.
- Parameter 2 (optional) specifies the device name or logical file from which the library is read. If this parameter is skipped, the library is read from logical file BI.
- Parameter 3 (optional) specifies the device name or logical file onto which the copied object module is written. The copied object module is followed by a file-mark record. If not specified, the default value is BO.

The syntax of the $DO command for the GET Procedure varies depending on the use of the object module's sequence-number or program-name. The following forms are both valid.

$DO GET,object-module-sequence-number[,input-file[,output-file]

$DO GET,object-module-program-name[,input-file[,output-file]

EXAMPLES

$DO GET,17,BI,BO
The 17th object module in the sequential library or logical file BI is copied to logical file BO.

$DO GET,PROG1
The object module under program-name PROG1 in the sequential library or logical file BI is copied to logical file BO.

A listing of the procedure follows:

```
$PROC GET,,BI,BO
$IFM %2,P=$ASS BI=BI;$ASS BI=%2
$REW BI
$IFM %3,P=$ASS BO=BO;$ASS BO=%3
$ASS L.O=L.O
$EXE LIB
GET %1
  EOF BO
EXIT
$NOTE END OF GET
```

APPENDIX A
ERROR MESSAGES

The error messages generated by Library Update are presented below. The presentation includes the error message, its meaning, and a recommended course of action.

CANNOT LIST/MODIFY A 32-BIT LIBRARY WITH LIBUP

Meaning: LIB for MAX IV cannot access a 32-bit library file.

Recommended Action: Exit LIB and invoke LIB32 and attempt the same directives.

CANNOT LIST/MODIFY A 16-BIT LIBRARY WITH LIB32

Meaning: LIB32 for MAX 32 cannot access a 16-bit library file.

Recommended Action: Exit LIB32 and invoke LIB and attempt same directives.

CANNOT RESTORE A 16-BIT SAVE FORMAT WITH LIB32

Meaning: LIB32 for MAX 32 cannot access a 16-bit library file.

Recommended Action: Exit LIB32 and invoke LIB and attempt same directives.

CANNOT SAVE A NULL MODULE

Meaning: The SAVE Directive cannot process a directory library which contains a null object module. That is, one containing only an end-record.

Recommended Action: Remove any null object modules through the REMOVE Directive. Re-enter the SAVE Directive.

CHECKSUM ERROR

Meaning: The checksum of an input record is not equal to its associated value in the record.

Recommended Action: If encountered during a SAVE, RESTORE, TEST, or SQUEEZE, enter /RESUME. These processes continue normally but the output AT the point of error is not valid. If encountered during any other directive, no recovery exists. LIB proceeds to the next directive.

DIRECTIVE NOT SUPPORTED

Meaning: The sequential library version of LIB/LIB32 was generated and a directory library directive was entered.

Recommended Action: Enter the appropriate sequential library directive.

DIRECTORY IN ERROR

Meaning: A directory library directive was entered that requires a directory on logical file UL. An invalid directory was found.

END MISSING

Recommended Action: The end of the object function code #7xxx is missing in an object module.

Recommended Action: Recompile/reassemble the object module after providing the function code.

END OF MEDIA

Meaning: A read was attempted at the end of a tape or disc.

Recommended Action: Rewind the device.

FILE MUST BE DISC

Meaning: The library being referenced must be on disc.

Recommended Action: COPY or RESTORE the library to a logical file assigned to disc.

FILE MUST BE DISC OR MAGNETIC TAPE

Meaning: The library being referenced must be either on disc or on magnetic tape.

Recommended Action: COPY or RESTORE the library to a logical file assigned to disc or magnetic tape.

FUNCTION ERROR

Meaning: An illegal function code is present in the current input object module.

Recommended Action: Recompile/reassemble the object module after correcting the function code.

ILLEGAL BINARY RECORD

Meaning: An input binary record contains an illegal binary header code (that is, not #03 or #07).

Recommended Action: Input a binary record that contains a legal header code (#03 or #07).

INITIALIZATION REQUESTED

Meaning: An initialization of the logical file BL has been requested through the INITIALIZE or RESTORE Directive.

Recommended Action: If LIB/LIB32 is resumed through the /RESUME Directive, the initialization occurs. The initialization may be avoided through the /ABORT Directive. Back up logical file BL to avoid loss of data.

INVALID FILE OPERATION

Meaning: An invalid logical file name was specified.

Recommended Action: Check ASSIGN Directives and verify which type of library (sequential or directory) is being accessed.

INVALID NAME

Meaning: An object module name was specified that does not start with an alpha character or that contains an illegal character.

Recommended Action: Rename the object module using from one to six RAD-code characters. The first character must be alpha.

INVALID SAVE FORMAT

Meaning: During a RESTORE, logical file BL was assigned to a library that was not in the required SAVE format.

Recommended Action: Check syntax of RESTORE Directive. Ascertain the need for the NODIRECTORY parameter within the directive.

MEMORY ALLOCATION FAILED

Meaning: The dynamic allocation of a sort buffer has failed, indicating insufficient memory resources.

Recommended Action: Cross-reference cannot be processed until additional memory becomes available.

MODULE HAS NOT BEEN PREVIOUSLY CATALOGED:

Meaning: The object module specified in a object RECATALOG Directive had module name no previous directory entry. LIB/LIB32 proceeds to catalog the object module.

Recommended Action: None.

MODULE NOT PRESENT

Meaning: The specified object module could not be located.

Recommended Action: Verify the name of the object module.

NAME PREVIOUSLY USED; object module name

Meaning: An existing object module is cataloged under the object module name specified in a CATALOG Directive. LIB does not catalog the new object module.

Recommended Action: Change the name of the new object module and re-enter the CATALOG Directive.

NAMELESS MODULE;CATALOGED AS $BLANK

Meaning: During certain processes, LIB uses the [program-name of the object module as the catalog-name. This warning message means that an object module with a program-name of spaces is being cataloged under the name $BLANK.

Recommended Action: None.

NO DIRECTORY

Meaning: LIB/LIB32 requires that a directory be present on logical file UL.

Recommended Action: Check ASSIGN Directives and verify which type of library (sequential or directory) is being accessed.

PARTITION SIZE NOT COMPATIBLE

Meaning: During a RESTORE, logical file UL was assigned to a disc partition smaller than the SAVEd library.

Recommended Action: Reassign UL to a larger disc partition.

SECTOR WORD SIZE NOT COMPATIBLE

Meaning: This condition is produced as a result of an invalid SAVE or RESTORE.

SEQUENCE ERROR

Meaning: The sequence number of an input record is invalid. This condition is also produced if the object module is empty.

Recommended Action: If encountered during a SAVE, RESTORE, TEST, or SQUEEZE, enter 'RESUME. These processes continue normally but the output at the point of error is invalid.

If encountered during any other directive, no recovery exists. LIB proceeds to the next directive.

SPACE IS INSUFFICIENT

Meaning: No space remains on the media assigned to a logical file while more object modules or records remain to be written.

Recommended Action: Assign the logical file to a larger space and reinitiate the process.

STATEMENT ERROR

Meaning: The directive entered is incorrect. This message reflects syntax errors in the directive. It may also be produced if a specified object module is not found.

Recommended Action: Correct the syntax then re-enter the directive. Verify that the object module is needed.

UNDEFINED FUNCTION CODE IN HEADER xy

Meaning: An internal error, or BI positioned to a non-standard binary record. The xy represents a two character Hex code.
x=0 undefined function code
y function code
x=1 unexpected EOF
y
x=2 function code size undefined
y function code

Recommended Action: Verify the BI, UL assignments and try again.

APPENDIX B
GENERATION PROCEDURES

The Released versions of MAX IV LIB and MAX 32 LIB32 have been generated to support both the Sequential and Directoried versions of LIB/LIB32. As of Revision H of LIB and Revision A of LIB32 there is also included an optional cross-reference command present in the Released binary version. If the user desires to build the XREF version of LIB/LIB32, [illegible], MAX IV [illegible]/MF [illegible] 15x-410200-000A.0 or later modules are required.

The user can acquire the source files for Library update. A Job Control generation procedure is included which assembles either version of LIB/LIB32.

LIBRARY UPDATE AS A FOREGROUND TASK

Instead of operating as an overlay of the batch task, LIB can be generated to run as an independent task under MAX IV or LIB32 can be generated to run as an independent task under MAX 32. *If this mode of operation is selected, LIB/LIB32 must be cataloged as a task on a directory disc file using the Task/Overlay Cataloger (TOC).

MAX IV AND MAX 32 ENVIRONMENTS

To generate a sequential library directive foreground task version of LIB/LIB32 under the MAX IV or MAX 32 Operating System, the following TOC Directives should be used:

```
TASK LIB (or TASK LIB32)
LOGFILE SC,BSC
LOGFILE WRK,BSW
LOGFILE CI,TY,,
LOGFILE LO,LP,,
LOGFILE BI,BSA,,
LOGFILE BO,BSB,,
LOGFILE 3
OPTIONS LO,HO,AO
CATALOG
```

* NOTE: LIB32 under MAX 32 must run as a 16-bit task. Therefore, TOC is used to catalog both LIB and LIB32.

APPENDIX C
SAMPLE LISTINGS

Sample listings for the LAF, LDI, LDE, LDS, SEA, SUL and XREF Commands are included below.

LAF P*******

OUTPUT

```
*A.0 MODCOMP 32-BIT LIBRARY UPDATE 03/11/85 17:08:21  PAGE 14
DIRECTORY LISTING OF $FM    $LO. =    LAST MODIFIED 03/05/85

MODULE          IDENTIFICATION          FILE     START    END      LENGTH

PRINTF.O        03/05/85    09:43       UL         28      35          8
PUTCHAR.        03/05/85    15:50       UL         71      72          2
```

LDE

OUTPUT

```
*A.0 MODCOMP 32-BIT LIBRARY UPDATE  03/11/85 17:08:21  PAGE 14
DIRECTORY LISTING OF $FM$  LO.=      LAST MODIFIED 03/05/85

MODULE          IDENTIFICATION          FILE     START    END      LENGTH

OPEN.O          03/05/85    08:46       UL         15      23          9
PRINTF.O        03/05/85    09:43       UL         28      35          8
CSTK.O          03/05/85    15:45       UL         52      53          2
MAIN.O          03/05/85    15.46       UL         54      60          7
CMAIN.O         03/05/85    15:47       UL         61      62          2
GETOPT.O        03/05/85    15:49       UL         63      70          8
PUTCHAR.O       03/05/85    15:50       UL         71      72          2
EXIT.O          03/05/85    15.51       UL         73      74          2
```

.DI

OUTPUT

```
*A.0 MODCOMP 32-BIT LIBRARY UPDATE  03 11/85 17:08:21  PAGE 14
DIRECTORY LISTING OF  $FMS   LD.=      LAST MODIFIED 03/05/85
MODULE       IDENTIFICATION        FILE    START   END     LENGTH

OPEN.O       03/05/85    08:46     UL        15      23         9
PRINTF.O     03/05/85    09:43     UL        28      35         8
CSTK.O       03/05/85    15:45     UL        52      53         2
MAIN.O       03/05/85    15:46     UL        54      60         7
CMAIN.O      03/05/85    15:47     UL        61      62         2
GETOPT.O     03/05/85    15:49     UL        63      70         8
PUTCHAR.O    03/05/85    15:50     UL        71      72         2
EXIT.O       03/05/85    15:51     UL        73      74         2
NUMBER OF ENTRIES CATALOGED =   8
NUMBER OF DIRECTORY ENTRIES USED =    16
NUMBER OF UNUSED DIRECTORY ENTRIES =     54
LARGEST NUMBER CONTINGUOUS UNUSED SECTORS =    345
TOTAL NUMBER OF UNUSED SECTORS =     375
TOTAL NUMBER OF SECTORS USED =      45
NEEDS TO BE COMPRESSED
```

LDS

OUTPUT

```
*A.0 MODCOMP 32-BIT LIBRARY UPDATE  03/11/85 17:08:21  PAGE 14
DIRECTORY LISTING OF  $FMS   LD.=      LAST MODIFIED 03/05/85


NUMBER OF ENTRIES CATALOGED =   8
NUMBER OF DIRECTORY ENTRIES USED =    16
NUMBER OF UNUSED DIRECTORY ENTRIES =     54
LARGEST NUMBER CONTINGUOUS UNUSED SECTORS =    345
TOTAL NUMBER OF UNUSED SECTORS =     375
TOTAL NUMBER OF SECTORS USED =      45
NEEDS TO BE COMPRESSED
```

SEARCH PRINTF

OUTPUT

```
*A.0 MODCOMP 32-BIT LIBRARY UPDATE  03/11/85 17:08:21  PAGE 14
DIRECTORY LISTING OF $F11$

PRINTF REFERENCE FOUND IN:
MODULE      PGM       INT       EXT      COM  REF

1     OPEN                    E         PRINTF
2     PRINTF    I               PRINTF
4     MAIN                    E         PRINTF
6     GETOPT                  E         PRINTF
```

SUL PRINTF

OUTPUT

```
PRINTF REFERENCE FOUND IN:
MODULE      PGM       INT       EXT      COM  REF

1     OPEN                    E         PRINTF
2     PRINTF    I               PRINTF
4     MAIN                    E         PRINTF
6     GETOPT                  E         PRINTF
```

XREF

OUTPUT

```
      16 INTERNALS IDENTIFIED.
      69 EXTERNALS IDENTIFIED.
      1 COMMONS IDENTIFIED.

 *** SORT BEGINS ***

DEMONSTRATION                                   03/11/85 17:08:21 PAGE 1

    T SYMBOL    MODULE      T SYMBOL    MODULE      T SYMBOL    MODULE

    E :BUFEN    GETOPT      E :CSTK     OPEN        E :CSTK     PRINTF
    E :CSTK     MAIN        E :CSTK     GETOPT      E :CSTK     EXIT
    I :CSTK     :CSTK       I :EXIT     EXIT        E :IOB      GETOPT
    E CMAIN     CMAIN       I CMAIN     MAIN        I CMAIN     CMAIN
    E CREX      MAIN        E CREX      EXIT        E CTERMI    GETOPT
    E CTOB      OPEN        E CUSERI    GETOPT      I ENVIRO    MAIN
    E ERRNO     OPEN        I ERRNO     MAIN        E EXIT      CMAIN
    E FDOPEN    GETOPT      E FGETS     GETOPT      E FINDMI    OPEN
    E FIX       MAIN        E FMOPEN    OPEN        E FOPEN     GETOPT
    E FREOPE    GETOPT      E FTELL     GETOPT      I GETOPT    GETOPT
    E GETS      GETOPT      E LDIVLL    PRINTF      E LMULLL    OPEN
    E LREMLL    PRINTF      E MAVR      MAIN        E MAIN      MAIN
    I MAXREG    MAIN        E MIOB      OPEN        I OPEN      OPEN
    E OPENFL    OPEN        I OPTARG    GETOPT      I OPTERR    GETOPT
    I OPTIND    GETOPT      I OPTOPT    GETOPT      E POPEN     GETOPT
    E PRINTF    OPEN        E PRINTF    MAIN        E PRINTF    GETOPT
    I PRINTF    PRINTF      I PRINTN    PRINTF      E PUTCHA    PRINTF
    E PUTCHA    PUTCHA      E REWIND    GETOPT      E SETBUF    GETOPT
    E SETBUF    OPEN        E STRCAT    MAIN        E STRCHR    OPEN
    E STRCHR    MAIN        E STRCHR    GETOPT      E STRCMP    OPEN
    E STRCMP    MAIN        E STRCMP    GETOPT      E STRCPY    OPEN
    E STRCPY    MAIN        E STRCSP    OPEN        E STRCSP    MAIN
    E STRLEN    OPEN        E STRLEN    MAIN        E STRNCA    OPEN
    E STRNCA    MAIN        E STRNCM    OPEN        E STRNCM    MAIN
    E STRNCP    OPEN        E STRNCP    MAIN        E STRPBR    OPEN
    E STRPBR    MAIN        E STRRCH    OPEN        E STRRCH    MAIN
    E STRSPN    OPEN        E STRSPN    MAIN        E STRTOK    OPEN
    E STRTOK    MAIN        E TEMPNA    GETOPT      E TMPFIL    GETOPT
    E TMPNAM    GETOPT      C ZERO      CMAIN
```

INDEX

Fold

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 3624 FT. LAUDERDALE, FL 33309

POSTAGE WILL BE PAID BY ADDRESSEE

MODULAR COMPUTER SYSTEMS
1650 W. McNAB ROAD
P.O. BOX 6099
FT. LAUDERDALE, FLORIDA 33310

Attention: TECHNICAL PUBLICATIONS, M.S. #85

Fold

MODCOMP

Please comment on the publication's completeness, accuracy, and readability. We also appreciate any general suggestions you may have to improve this publication.

If you found errors in this publication, please specify the page number or include a copy of the page with your remarks.

Your comments will be promptly investigated and appropriate action will be taken. If you require a written answer, please check the box and include your address below.

☐

Comments: _____

Manual Title _____

Manual Order Number _____ Issue Date _____

Name _____ Position _____

Company _____

Address _____

_____ Telephone ( ) _____

Corporate Headquarters
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel. (305) 974-1380, TWX 310-372-7937

International Headquarters
MODULAR COMPUTER SERVICES, Inc., The Business Centre, Molly Millars Lane, Wokingham, Berkshire, RG11 2QJ, UK, Tel. 0734-786936, TLX 851849149

Latin American Sales Headquarters
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 6099, Ft. Lauderdale, FL 33310, Tel. (305) 975-6582, TLX 3727652

Canadian Headquarters
MODCOMP Canada, Ltd., 400 Matheson Blvd. East, Unit 24, Mississauga, Ontario, Canada L4Z 1N8, Tel. (416) 890-0666, TELEX 06-961279

SALES & SERVICE LOCATIONS THROUGHOUT THE WORLD

Printed in the U.S.A.